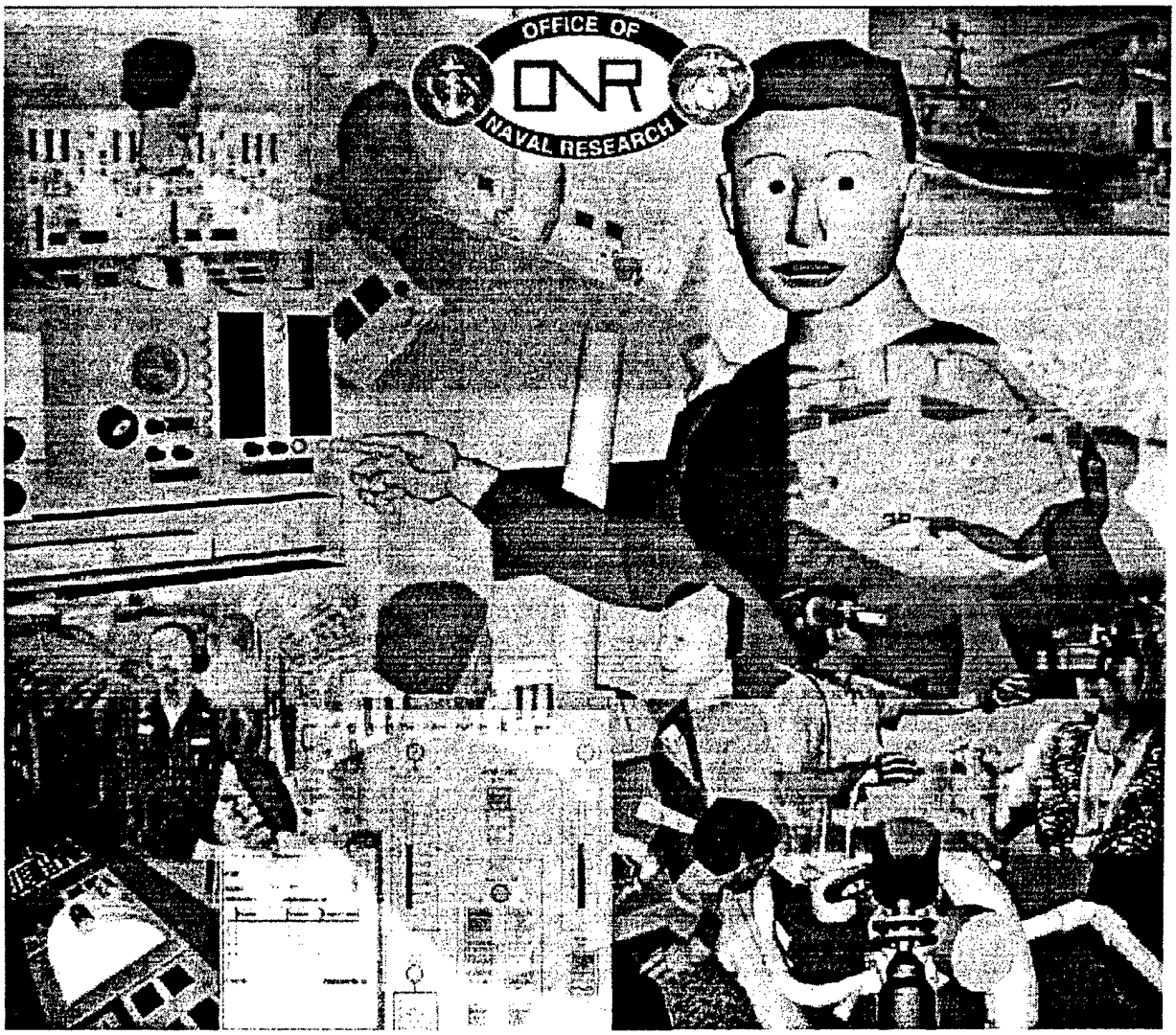
Virtual Environments for Training
Final Report



Virtual Environments for Training

Final Report

Bound Final Report
LMMS Doc. Control #P499363
Virtual Environments for
Training Final Report

TECH POC: Randy Stiles
650.354.5256 Randy.Stiles@
LMCO.COM

Prepared for
Office of Naval Research
Contract N00014-95-C-0179
(CDRL A002 Final Report)

April 1999

Submitted by
Randy Stiles

Advanced Technology Center
Lockheed Martin Missiles & Space
O/L922 B/255, 3251 Hanover St.
Palo Alto, CA 94304-1191

DISTRIBUTION STATEMENT AUTHORIZATION RECORD

Title: Virtual Environments for Training

Authorizing Official: Helen Gigley

Agency: ONR Ph. No. (703)696-0407

☐ Internet Document: URL: _____
(DTIC-OCA Use Only)

Distribution Statement: (Authorized by the source above.)

☒ **A**: Approved for public release, distribution unlimited.

☐ **B**: U. S. Government agencies only. (Fill in reason and date applied). Other requests shall be referred to (Insert controlling office).

☐ **C**: U. S. Government agencies and their contractors. (Fill in reason and date applied). Other requests shall be referred to (Insert controlling office).

☐ **D**: DoD and DoD contractors only. (Fill in reason and date applied). Other requests shall be referred to (Insert controlling office).

☐ **E**: DoD components only. (Fill in reason and date applied). Other requests shall be referred to (Insert controlling office).

☐ **F**: Further dissemination only as directed by (Insert controlling DoD office and date), or higher authority.

☐ **X**: U. S. Government agencies and private individuals or enterprises eligible to obtain export-controlled technical data in accordance with DoD Directive 5230.25.

NOTES: _____

Joyce Keith
DTIC Point of Contact

4 JAN 2000
Date

LOCKHEED MARTIN

Lockheed Martin Missiles & Space
Advanced Technology Center
3251 Hanover Street, Palo Alto, California 94304-1191

Monday, April 26, 1999

Program Officer
Code 342, Helen Gigley, Ph.D.
Office of Naval Research
800 North Quincy Street
Arlington, VA 22217-5660
Ref: N00014-94-C-0179
(703) 696-0407

Subject: Virtual Environments for Training Final Report

Dear Dr.Gigley;

This document constitutes the Virtual Environments for Training (VET) Final Report. This report describes reusable results, identifies problems common to similar efforts, notes useful ideas, and provides suggestions for further related activities. This report also serves as a record of tasks accomplished to satisfy the proposed VET work.

The final report sections were created using input from Lewis Johnson, Ph.D., and Jeff Rickel, Ph.D. at USC Information Sciences Institute, Marina Del Rey, CA, and Allen Munro Ph.D., and Quentin Pizzini, Ph.D. at USC Behavioral Technology Labs, Redondo Beach, CA. The research and development described in this final report is the collective effort of these people: Rich Angros, Craig Hall, Ph.D., Carol Horwitz, Lewis Johnson, Ph.D., Mark Johnson, Laurie McCarthy, Mihir Mehta, Allen Munro, Ph.D., Quentin Pizzini, Ph.D., Jeff Rickel, Ph.D., Erin Shaw, Ph.D., Randy Stiles, Sandeep Tewari, Marcus Thiebaux, and Josh Walker.

We really appreciated the opportunity to work for the Office of Naval Research on the Virtual Environments for Training contract. I feel that our team has accomplished the objective that we proposed for the VET Focused Research Initiative: "*to develop an instructional system that integrates the design, development, delivery and evaluation of training curricula within a comprehensive virtual environment*".

Respectfully,

Randy Stiles

Randy Stiles
Program Manager, Virtual Environments for Training
O/L922 B/255

Randy.Stiles@LMCO.COM
http://vet.parl.com/~vet/
(415)354-5256, fax: (415)354-5235

Table of Contents

Table of Figures

A Executive Summary

This document is the Virtual Environments for Training (VET) final report. VET, funded by the Office of Naval Research (ONR), began in October 1995 and completed in February 1999. VET focused on immersed instruction, culminating with the successful development of the Training Studio prototype (see Figure 1), an authorable system for team instruction in a virtual environment. Several key advances have resulted from the VET program:

- Pedagogical agents that can act as student mentors or team members during training
- Component-based instructional systems
- Authorable virtual environment interaction applicable across domains

In the executive summary of our 1994 VET proposal, we proposed to:

> *"develop an instructional system that integrates the design, development, delivery, and evaluation of training curricula with a comprehensive virtual environment."*

Using our Training Studio system, authors can design and develop intelligent instruction for delivery to students, and evaluation of their performance, in an immersed virtual environment. Thus, the VET Training Studio is an example of an integrated, working VE system for constructing, managing, and interacting within virtual environments, satisfying these challenges:

- *The most immediate challenge at hand is one of integrating the existing technology into a working system, along with other elements of VE construction software.* in 1995 National Research Council study on virtual environment technology (Durlach, 1995)
- *There is a need for software infrastructure and tools for constructing, managing, and interacting within virtual environments.* Barfield and Furness, discussing Virtual Environment Interfaces (Barfield, 1995)

The Training Studio includes a broad range of capabilities applicable to many types of training in virtual environments. First, it includes extensive support for authoring. This includes the ability to import CAD models for rapid construction of the 3D graphical models of the virtual environment, create the simulation behaviors of the environment, define the instructional objectives,

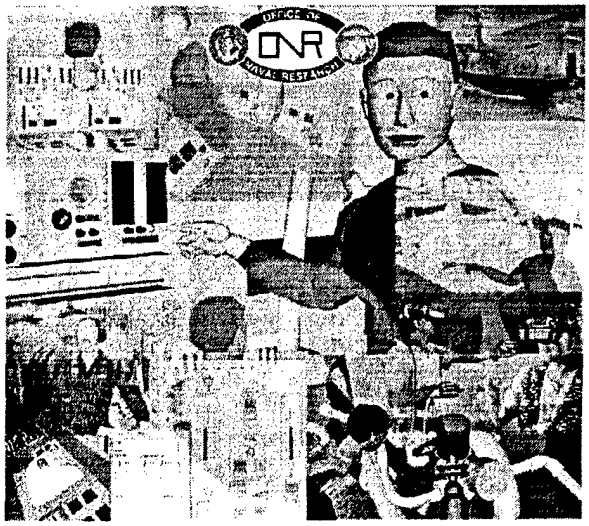
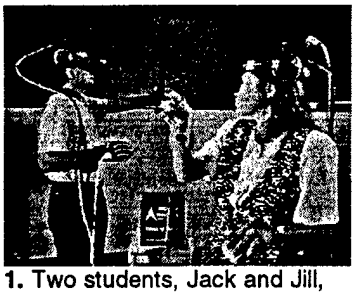


Figure 1. The Training Studio software is a progenitor for a "HoloDeck" where people engage in immersed interaction with simulations of other people and scenes

and construct a wide variety of training exercises. The authoring approach is designed to ensure the development and maintenance of high quality instruction at significantly lower cost than could be achieved using conventional one-time development methods. Second, it supports delivery of instruction in virtual environments, including both individual and team training. Training exercises can range from simple drill-and-practice familiarization lessons to more complicated training scenarios involving simulated shipboard casualties. Moreover, the training can include intelligent agents that appear in the virtual world as virtual humans and serve as instructors or teammates when human instructors and teammates are unavailable; such agents provide exciting new possibilities for automated instruction and interaction with the student that are not possible with more traditional computer-based instruction.

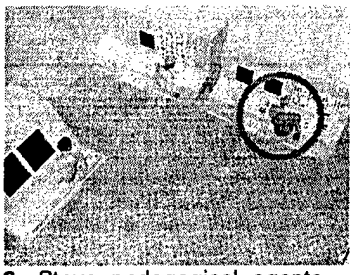
To describe the VET Training Studio in this final report, we discuss the operational, system and technical views for the VET Training Studio. During the VET effort, we developed and implemented an operational view of how a VE training system should be used, a system view of how the components work together, and a technical view of the standards and conventions necessary for our training authoring focus. In the system and technical view sections, we relate the significance of our work on virtual environment interaction, simulation-based training, and pedagogical agents.

Table 1. Training Studio Instructional Delivery Thumbnail Description (See section C.6.2)

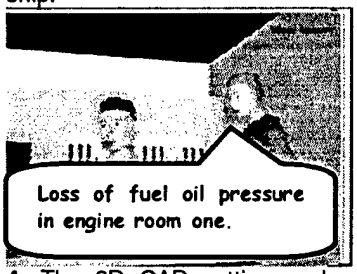


1. Two students, Jack and Jill, learn their roles for casualty control procedure on board a ship.

2. Each student wears a head-mount display that immerses them in the 3D CAD models for the ship.

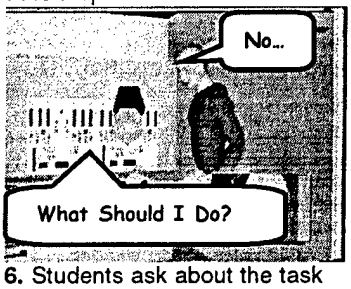


3. Steve pedagogical agents serve as their team-mates or tutors. Jack's tutor is the blue PACC operator circled above.

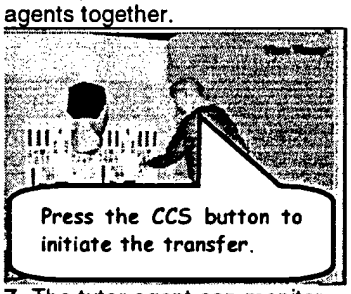


4. The 3D CAD setting and equipment simulations are used by the students and the agents together.

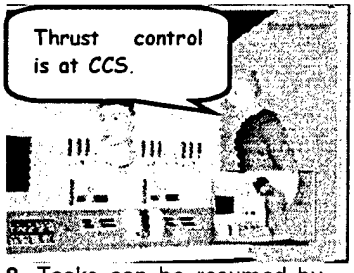
5. Students talk to the tutor to try doing part of the task. Speech acts are part of the instruction.



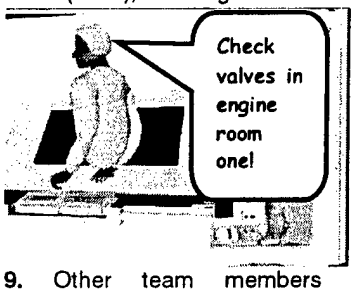
6. Students ask about the task using speech recognition. Jill's tutor is in the engine room (inset), awaiting tasks.



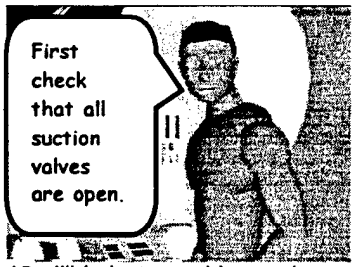
7. The tutor agent can monitor the student's view and actions in the world, and the effects are promulgated in the simulation.



8. Tasks can be resumed by the tutor.

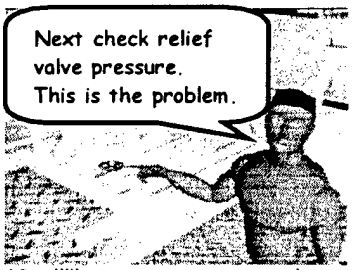


9. Other team members (agents) communicate and act out their team roles.



10. Jill is instructed in another part of the ship, the engine room.

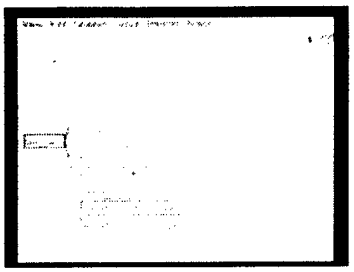
11. Tutors lead students through equipment spaces, this is an aspect of spatial instruction.



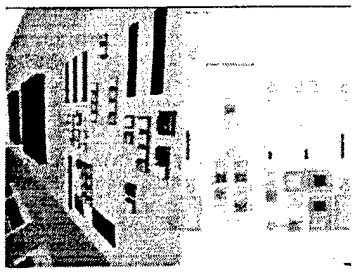
12. Jill's tutor agent uses the equipment to show her the procedure to reset relief valves.

Table 2. Training Studio Instructional Development Thumbnail Description (see section C.6.1)

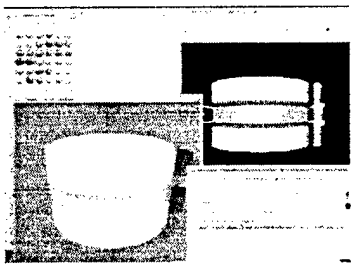
1. First authoring task is understanding the domain, in this case mentors oversee team members.



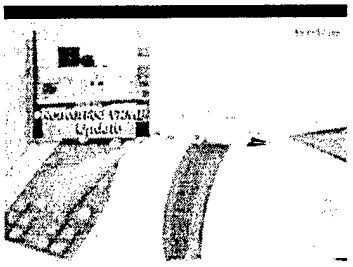
2. Next authoring task is understanding course structure, and realizing this in VIVIDS.



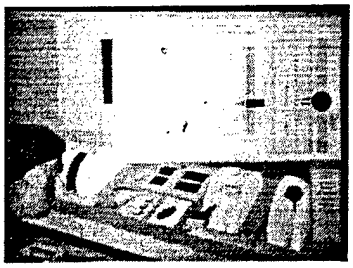
3. The equipment simulations are authored in VIVIDS in 2D, for eventual use in 3D.



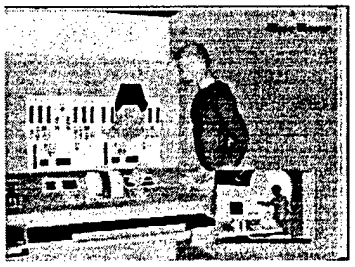
4. 3D equipment is imported from CAD to VRML, and VRML COTS authoring tools are used to define interaction.



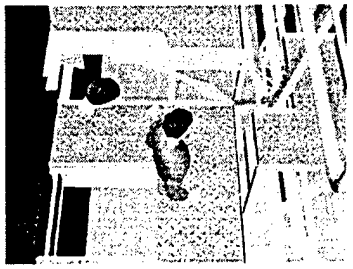
5. Interaction immersed, and in shared settings is tested in Vista, and 3D models are modified to support interactive rates, and behaviors.



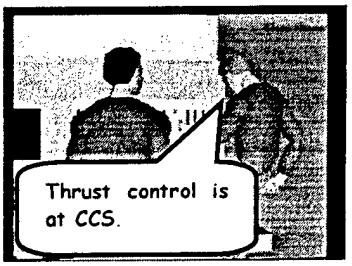
6. The 2D equipment simulations in VIVIDS are linked to the 3D behaviors in Vista using TScript message updates.



7. The Steve agent is defined based on the task procedures, and equipment simulations in VIVIDS.



8. Steve agents use waypoint information to move around the spatial setting to different equipment.



9. Team roles are refined, and speech acts are modeled between team members. Task instruction is tested for robustness.

B Introduction

This document is the final report for the Virtual Environments for Training (VET) program, funded by the Office of Naval Research (ONR). The VET program's period of performance started in October 1995, and completed in February 1999, culminating in the successful development of the Training Studio prototype, an authorable, component-based system for team instruction in a virtual environment.

Virtual environments are three-dimensional (3D) scenes where participants perceive themselves as being inside a 3D scene. Virtual environment software typically draws 3D scenes many times a second so that motion in the scene in relation to head movements is perceived as fluid, and stereoscopy and parallax give the perception of solid objects. Additionally, participants can manipulate objects in the scene, providing the perception that the participant can affect change in the scene.

The Training Studio includes a broad range of capabilities applicable to many types of training in virtual environments. First, it includes extensive support for authoring. This includes the ability to import CAD models for rapid construction of the 3D graphical models of the virtual environment, create the simulation behaviors of the environment, define the instructional objectives, and construct a wide variety of training exercises. The authoring approach is designed to ensure the development and maintenance of high quality instruction at significantly lower cost than could be achieved using conventional development methods. Second, it supports delivery of instruction in virtual environments, including both individual and team training. Training exercises can range from simple drill-and-practice familiarization lessons to more complicated training scenarios involving simulated shipboard casualties. Moreover, the training can include intelligent agents that appear in the virtual world as virtual humans and serve as instructors or teammates when human instructors and teammates are unavailable; such agents provide exciting new possibilities for automated instruction that are not possible with more traditional computer-based instruction.

Many obstacles exist for team training in a virtual environment. Students must perceive the scene and the actions of other team members in common. Often, instruction of teams can be hindered by slower members of the team, and because of conflicting schedules, it can be difficult to assemble team members and resources (such as dedicated simulators or actual equipment) for instruction. The Training Studio addresses these obstacles.

To describe the VET Training Studio in this final report, we discuss the operational, system, and technical views for VET. During the VET effort, we developed and implemented an operational view of how a VE training system should be used, a system view of how the components work together, and a technical view of the standards and conventions necessary for our training authoring focus. In the system and technical view sections, we relate the significance of our work on virtual environment interaction, simulation-based training, and pedagogical agents.

B.1 Program Final Report

This final report has describes reusable results, identifies problems common to similar efforts, notes useful ideas, and provides suggestions on further related activities. This final report also serves as a record of tasks accomplished to satisfy the proposed work.

B.2 Program Team

The Virtual Environments for Training team consists of Lockheed Martin Advanced Technology Center for virtual environment interaction, USC Behavioral Technology Laboratories for instructional simulation, USC Information Sciences Institute for pedagogical agents, and Air Force Research Lab at Brooks, AFB for outside formative evaluation.

B.3 Program Objectives

In the executive summary of our VET proposal, we proposed to *develop an instructional system that integrates the design, development, delivery, and evaluation of training curricula with a comprehensive virtual environment.* The emphasis was on an integrating architecture to realize effective virtual environment training.

We have realized our proposed objectives for training in a Virtual Environment. The primary statement of work items in our original proposal

emphasized authorable virtual environments in which team training could be realized:

- Conduct innovative research into team training and modeling in a virtual environment including efficient distribution mechanisms and representations, and advanced models of team interaction in a virtual environment.
- Develop an instructional design approach appropriate to authoring courses for intelligent tutoring systems integrated with a virtual environment.
- Develop and refine the human-computer interface approach for development and delivery of instructional content in a virtual environment.

Our research into *team training and modeling* in a virtual environment has been successful. Using the Steve pedagogical agent as team members and student mentors, we can model team interactions and instruct students on their role in a team.

As a basis for all our efforts, we developed an *instructional design approach* for collaborative authoring of intelligent tutoring in a virtual environment that consists of equipment simulations, spatial representations, and team procedures and roles.

Our *human-computer interface approach* encompasses interaction with the virtual environment scenes, a set of spatial services necessary for understanding the student's actions, and detailed spatial feedback to the student for effective instruction.

C Methods & Approach

This section outlines the methods and approaches that evolved from our proposal, and during the VET effort.

C.1 Component Systems

Ours was a problem of integrating capability at the right level. In our systems view, we decided to reduce technical risk by using a component architecture, where the major components were virtual environment interaction (Vista), instructional simulation (VIVIDS), and pedagogical agents (Steve). This reduced risk significantly because the components were mature and debugged, and source code did not have to be merged. Development was carried out in parallel and components were integrated as part of a spiral development process. Our team tested component changes and improvements at system level test and integration meetings. At the start of the VET effort, this component-based approach was novel to the intelligent tutoring systems research community.

C.2 Multi-modal Interaction

A key aspect of our approach from the beginning of VET has been to deliver training in a fully immersed setting. Immersed by our definition did not mean a 3D image on a screen. Full immersion meant that the person perceived themselves to be inside the spatial scenes we presented.

Using virtual environments for training poses a number of challenges and opportunities for human-computer interaction. Opportunity exists in the area of multi-modal interaction, where multiple modes, such as speech and spatial display, are used concurrently to provide information to the student, and to get information from the student. Beyond the promise of transferring training in a virtual 3D setting to the real setting, there are advantages to instruction that combines modes of interaction. When modes for 3D display, 3D manipulation, 3D head motion, and dialog systems (speech generation from text, and speech recognition) are combined with object simulation, a property known as spatial dialog emerges. The context of speech acts is provided by what the participant is looking at, near to, or touching. It becomes possible to ask questions previously ambiguous, and to carry out actions in the spatial setting easier.

C.3 Authorable Systems

From the outset of the VET effort, we emphasized that systems should be built with authoring in mind to ensure their widespread use and effectiveness.

Authoring the spatial setting for instruction in a virtual environment involves the geometry of objects, the physical dynamics of objects, and the behavior of objects. Based on our technical view of the Training Studio, we decided spatial settings would be authored in VRML, where commercial software could be used to import geometry, modify object geometry, and specify object interactions in an effective manner. Since we had an effective means of authoring complex object behavior for objects using VIVIDS, we arrived at a systems view where object behavior is authored using VIVIDS, and the geometry and motion constraints are authored in commercial VRML authoring packages. In Vista, we focused on the capability to load VRML object geometry, realize VRML object interaction while immersed, and update object state using VIVIDs behaviors.

Although Steve is a complex intelligent system, it is designed to support authoring by people with limited programming experience by using artificial intelligence methods. Although the decision making component of Steve is implemented in Soar (Laird et al 1987), Steve does not require the courseware developer to know how to program in Soar. Instead, Steve's task knowledge is specified using a high level plan language, in which plans are built from a library of primitive operations. An authoring interface has been developed for integrating such plan descriptions with VIVIDS simulation models; once Steve is provided with a suitable task plan and is linked to the VIVIDS model, Steve is capable of interacting with the VIVIDS model to perform asks, monitor students performing tasks, etc. An extension to Steve called Diligent was created for creating task plans. It employs a programming-by-demonstration approach, where the instructor demonstrates how to perform the task and then Diligent attempts to generalize the demonstration to produce a complete task plan. Diligent also provides a graphical editing interface for modifying task plans.

C.4 Task-based Instruction

As Hill and Johnson (1995) argued, simulation-based instruction without guidance can be inefficient and error prone. Students working with a simulation may discover that they do not know what to do, or fail to recognize when they have made a mistake. The VET project has postulated that animated agents are a natural means for providing such guidance. An animated human figure can guide the students through the virtual space, show students what to do, and direct their attention to important elements of the virtual environment. They can interact with the students via natural face-to-face dialog.

C.5 Team Training

The VET project has postulated that team interaction can be more effectively captured and modeled using an immersive virtual environment. Team member locations, speech acts, physical gestures, equipment manipulations, and viewpoints provide additional means of communication beyond conventional computer displays, and we have used these with some success.

Conventional team training is a very labor-intensive activity. A typical team training exercise at the Navy's Great Lakes Training Center, for example, requires that a team of trainees be assembled, together with a team of instructors who guide the trainees through the exercise. Synthetic agents can significantly reduce the labor requirements for team training exercises. In VET team training exercises, each participating student can have a Steve virtual instructor guide them through the exercise and answer routine questions, freeing human instructors to focus their attention on critical problems. Steve agents can also play the roles of missing team members, allowing individual students and small groups to practice their skills without assembling large teams.

C.6 Operational View

In this section, we provide an operational view of the VET Training Studio. An operational architecture view is a description of the tasks and activities, operational elements, and information flows required to accomplish or support an operation. This section provides an operational view of the Training Studio as it is used for instructing engine room casualty control procedures, from both an instructional development and an instructional delivery perspective.

Visual descriptions of Training Studio Instructional Development and Instructional Delivery, meant to accompany the following text descriptions, are available in section A (see Table 1, Table 2).

C.6.1 Instructional Development

The usage model for the completed Training Studio system is important as a guide to the process of authoring shipboard training using the Training Studio system. Note that while the system has not yet been deployed in this manner, the usage model provides a way of understanding why Training Studio capabilities were developed, and their intended use together.

For delivery of training material developed with the Training Studio we started with an idea of how training material could be delivered and used as part of Navy operations - exchanged over the Internet or a similar closed network system. The

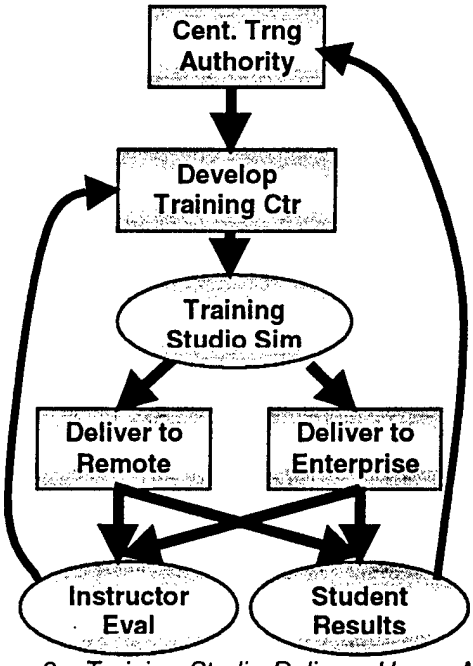


Figure 2. Training Studio Delivery Usage Model emphasizes internet (WWW) distribution

delivery approach can be adapted to other services or organizations which are dispersed.

In the delivery usage model (see Figure 2), a demand for training in a particular topic comes from a central training authority which is tasked with prioritizing and funding training efforts. This request is handled at a Training Center which has the appropriate hardware, software, and personnel to develop the training material. A course using the Training Studio is developed, and fielded to inland reserve areas and out to ships at sea using the World Wide Web protocols on the Internet or a similar but secure network system. The ships and inland reserve sites involved have equipment sufficient to run the Training Studio software, and one or more people who are responsible for updating/maintaining the software and hardware.

After delivery of the Training Studio simulations, students use the system to train, and people at the site with experience in the subject domain evaluate the initial courses and simulation for effectiveness (these may be actual instructors or specialists). The results associated with each student in the course are relayed to the central training authority over the same network, and the results of instructor evaluations are returned to the Training Center for use in refining the system (the student's evaluations of the training material could be used there too). Unlike centers using traditional curricula, the training center described here could act on the requested changes and update remote sites using the curricula fairly quickly.

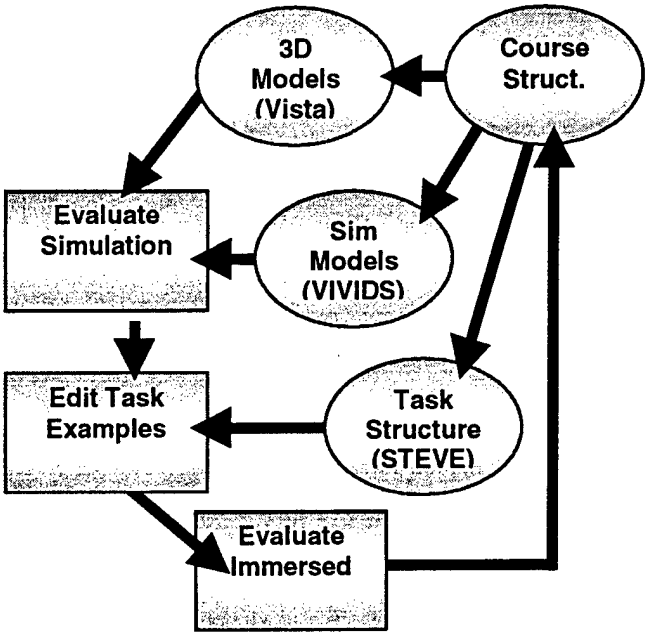
We have taken steps in the system development to support distributing the training simulations using the World Wide Web, allowing distribution of 3D models and VIVIDS simulation courses. Given the large amount of data associated 3D models and simulations, network transfer is somewhat of a technical challenge, but we have taken measures such as caching to address transfer of 3D data. The Training Studio does not at this point have support for relaying instructor evaluations of the system or the student scores in training back out of the system. This feedback could be developed fairly easily using current WWW resources.

The actual development of a course for a given domain starts at a Training Center by defining the Course Structure based on approved procedures in reference works, subject matter expert (SME) input, etc. (see Figure 3). Here the instructional developers get an idea of what material needs to be covered, what simulations may be helpful, what the tasks in the domain may be, and which 3D models will be needed. This gives the developers a shopping list and an initial course of action.

Initial training simulations are developed in 2D using VIVIDS and at the same time members of the development group assess existing 3D model resources and obtain these. Soon after initial simulations are developed, they are modified to update the 3D scenes in Vista using TScript messages. At each stage, the simulation is evaluated by the development group to see if it matches real system behavior in the areas significant to training transfer.

When the simulation is detailed enough to support a given task in a 3D setting, the instructional developers use Vista and the rest of the Training Studio software to provide task examples. Simpler examples of tasks can be modeled using VIVIDS, and more complex task examples where the tutoring system may need to provide spatial,

Figure 3. Training Studio Development Usage Model allows concurrent development



visual explanation of task steps to the student will be captured by the Steve software.

Finally, experts in the domain evaluate the course immersed in the virtual environment, to see that necessary aspects of the tasks are captured faithfully. If improvement is needed, this feeds back into the course structure, and then again into 3D models, simulation models, and task structure for the lessons.

Several types of simulation models must be built during domain development. In our usage model, a good portion of support for capturing the simulation models themselves is present in VIVID,

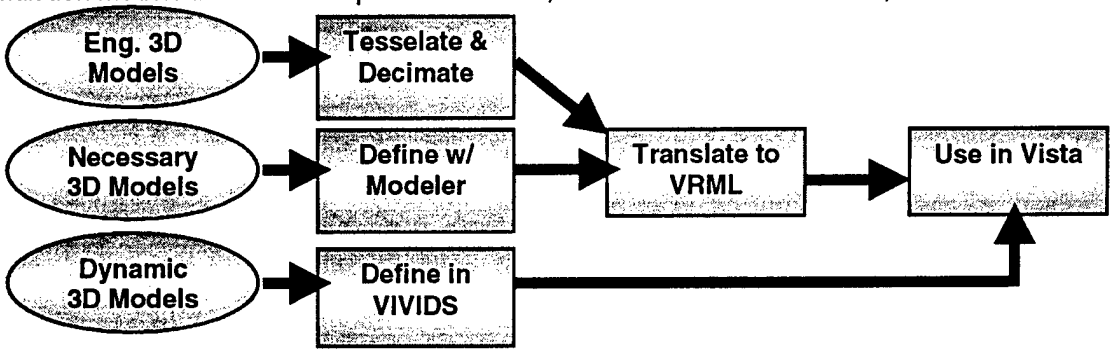


Figure 4. Training Studio Usage Model for 3D Model Development emphasizes real-time immersed interaction with VRML 97 scenes

and we have built support in the Steve software for capturing task models using a programming-by-example approach. The 3D model data used in the virtual environment is one area that can be labor intensive, and where existing data may be available to re-use. So in our usage model we have further elaborated on how we see 3D model data entering into system usage (see Figure 4).

The usage model for using 3D models recognizes three forms of models: engineering (CAD) models and already existing visual simulation models which can be purchased, the set of necessary 3D models not already existing, and those 3D models which, because of the nature of the domain or its explanation, must be dynamic (mutable while using the virtual environment).

For existing engineering models, it is often the case that the polygon count is too large for real-time interaction in a virtual environment. This is because engineers often export the model tessellated polygons without tessellating the models for lower polygon counts. The ideal place to reduce polygon counts is when exporting models using the CAD modeler originally involved in during the export of original models. Often the models can be re-tessellated to a much lower polygon count using the CAD data and the original modeler.

However, most often we are not that lucky. Polygons in complex engineering models can be reduced using decimation, whereby parameters are set such as length of the polygons or area relative to their importance during decimation, and vertices for less important polygons are removed to create fewer polygons. One of the easier-to-use commercial packages for polygon reduction is available from Silicon Graphics in their CosmoWorld software, a tool for editing and optimizing VRML scenes. This is what we use in our domain development. Research in other advanced methods for decimation has been funded by the Office of Naval Research, and could fill this place in the process (Renze and Oliver 1996).

At times a project will have to resort to generating their own special 3D models. Any necessary 3D models that can't be reused must be built using a modeling system. We have found MultiGen Flight and the SGI Inventor tool set to be useful in this regard. There are many commercial modeling systems available, and for our purposes, if their output can be translated to VRML, they can be used. We recently finished a utility that allows models loaded into Vista to be saved in the Inventor 3D model format. The 3D models can easily be taken from there to VRML. This was done in support of the "translate to VRML" step in the usage model. Some notable 3D file formats that Vista loads and can now be saved as Inventor (or VRML with the pf2wrl utility) include: the S1000k DoD terrain format, AutoCad DXF and 3D Studio, Wavefront, Lightscape, Multigen Flight, Coryphaeus Designer Workbench.

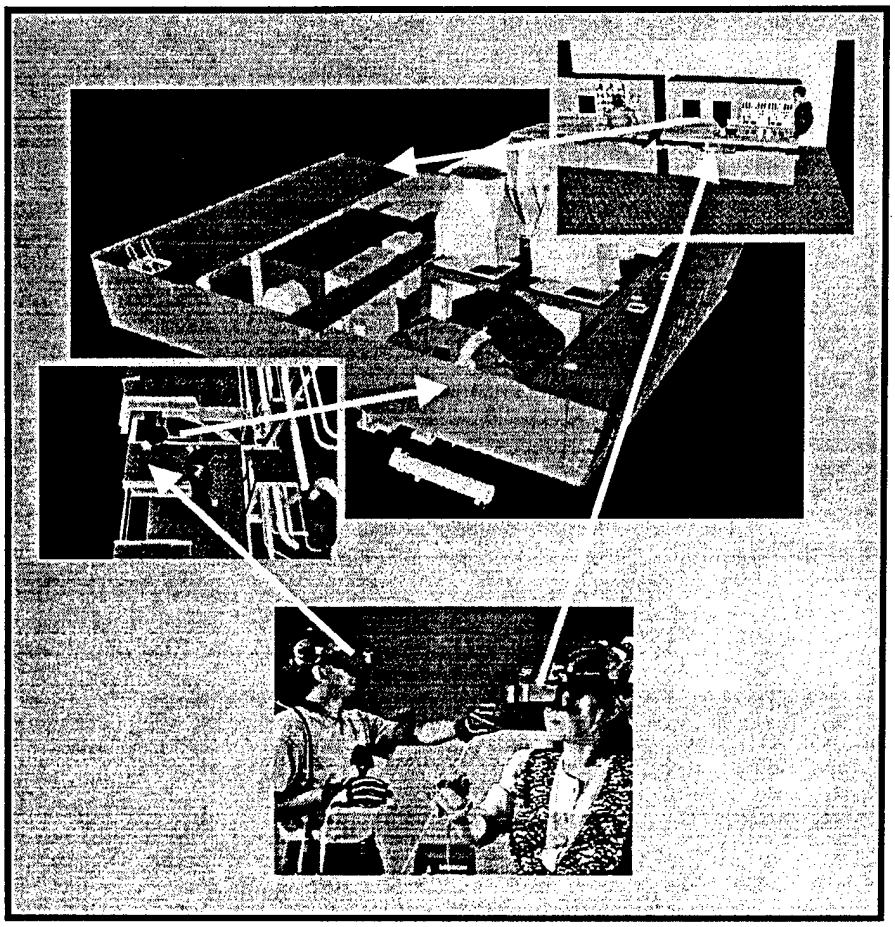


Figure 5. Immersed Instructional Delivery enables training for different team roles, and locations

The third category of 3D models, dynamic models, are those whose geometry must change live during the simulation usage. This occurs often during explanation, where for example a polygon representing the arc of an angle must change to represent a changing angle value, or a visual line showing a link between two objects must change when the objects move, etc. These models are best defined using the graphical primitives in Vista, from the simulation where their values are calculated (such as VIVIDS). Vista provides a large set of graphical primitives such as cubes, etc. for dynamic update of their appearance independent of modeler software, and we support modifying VRML geometry on the fly by using external Tscript messages, or VRML interpolators.

C.6.2 Instructional Delivery

To illustrate the operation capabilities of the Training Studio for instructional delivery, consider a training scenario in which two students, Jack and Jill, are learning their roles in handling a loss of fuel oil pressure in one of the gas turbine engines aboard a ship (see Table 1).

Jack is serving as the Propulsion and Auxiliary Control Console (PACC) operator, and Jill is in the engine room. Each student is wearing a head-mounted display that provides a three-dimensional view of their shipboard surroundings. As they move or turn their head, their view changes accordingly. Each student is assigned a Steve agent as their tutor. In addition, three other Steve agents serve as their teammates: one serves as the Engineering Officer of the Watch (EOOW), one serves as the Electric Plant Control Console (EPCC) operator, and one serves as the Shaft Control Unit (SCU) operator. Each person's head-mounted display is equipped with a microphone (for speaking to agents) and earphones (through which the person can hear agents speak, as well as sound effects from the virtual environment).

Jack's tutor looks at him and introduces the scenario: "Let me show you how to handle a loss of fuel oil pressure. First, when you detect it, inform the EOOW." Looking over at the EOOW, the tutor continues, "We have a loss of fuel oil pressure in engine room one." The EOOW nods in acknowledgment and passes the message on to the engine room. Jack's tutor leads him over to the normal stop button, points at it, and says, "First, press the normal stop button to stop the turbine." The tutor presses the button, and Jack

watches its indicator light up and the engine's power lever angle go to idle. "I will now transfer thrust control to the central control station," the tutor informs Jack. Jack, believing that he remembers the procedure, says, "Let me finish." "Okay, you finish," replies the tutor, who shifts to monitoring Jack's performance of the task.

Jack steps forward to the console and presses the wrong button. "No," the tutor comments while shaking his head. Jack, suddenly less sure of himself, asks, "What should I do?" The tutor replies, "I suggest that you press the CCS button to initiate the transfer." Jack presses the button and his tutor nods approvingly. As a result of Jack's action, the CCS button blinks on both the PACC and the SCU. The SCU operator, in the engine room, presses the blinking CCS button on his console to complete the transfer, and the button stops blinking and remains illuminated on both consoles.

Jack looks over at the EOOW and says, "Thrust control is now at CCS." The EOOW nods in acknowledgment and instructs the EPCC operator to switch to generator one. Jack watches as the operator pushes a series of buttons and informs the EOOW when the switch is complete. Now the EOOW commands the engine room to investigate the cause of the casualty.

Upon receiving the command, Jill's agent says, "Let me show you how to check for the cause of the loss of fuel oil pressure. First, check that all suction valves are wide open. A partially closed valve in the suction line can increase the suction lift above the pump's capabilities." The agent guides Jill around the engine room and shows her the location of the valves. They check each one, but all are already wide open.

Next, the agent leads Jill over to the relief valve. As she gets close, she can hear the sound of the oil passing through the valve. "Next, check the relief valve set pressure. As you can hear from the sound of the oil passing through the valve, the set pressure is too low, causing the loss of fuel oil pressure." The agent shows Jill how to reset the relief valve lifting pressure, then reports back to the EOOW, "The cause of the casualty has been determined and corrected."

Although this scenario does not illustrate all of the capabilities of the Training Studio, it highlights some of the most important. Jack and Jill cohabit a virtual mockup of their work environment, along with Steve agents that serve as their tutors and teammates. They can navigate around the environment to learn the location of relevant equipment, often under the guidance of their agent tutor. Agents and students can manipulate objects in the virtual world and see their visual and auditory effects. Finally, they can collaborate with each other, as well as their agent tutor and teammates, to practice realistic training scenarios.

D Results

This section describes significant and reusable results from the VET effort, encompassing the system view and technical view.

D.1 System View

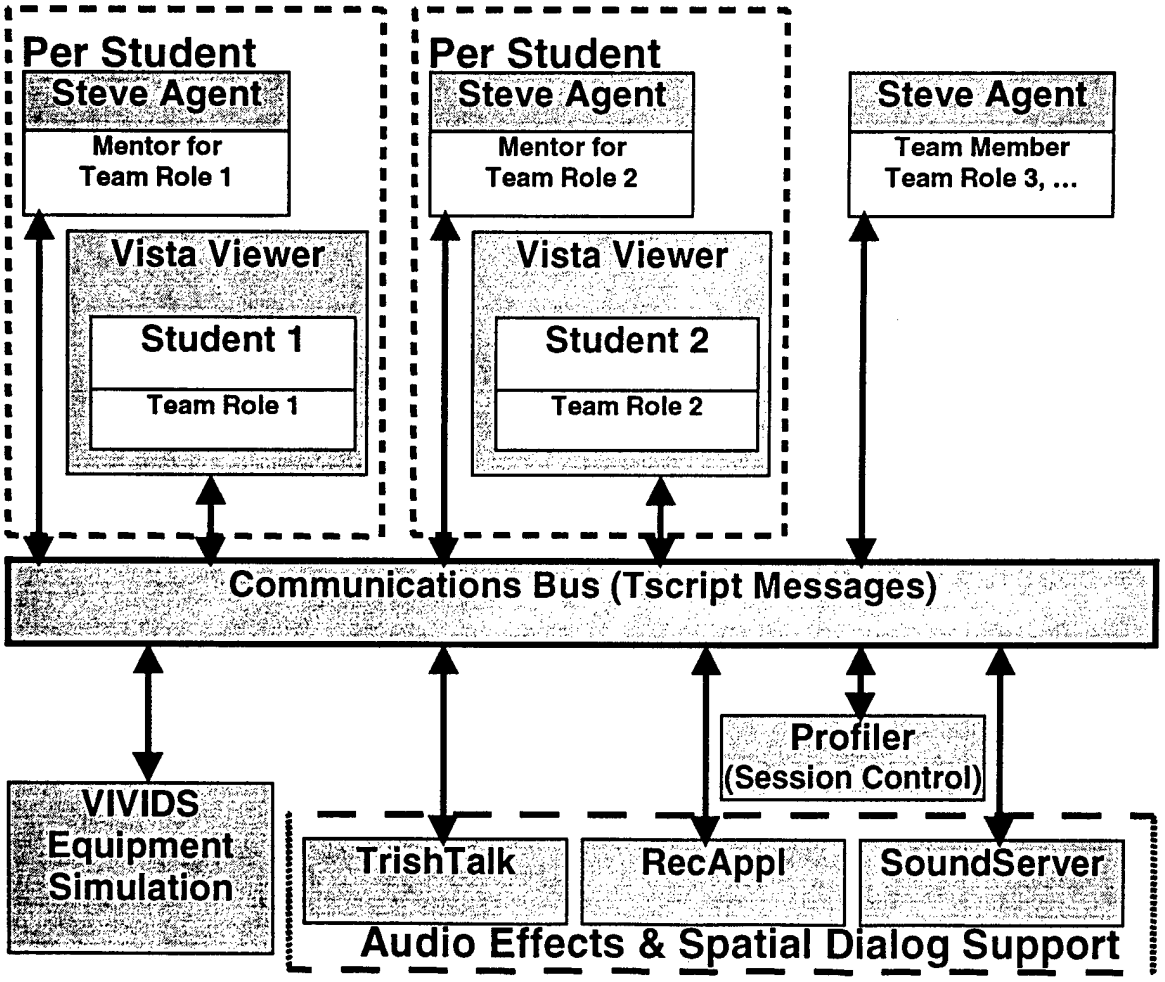
The systems architecture view is a description, including graphics, of systems and interconnections providing for, or supporting, system operation. We describe the Training Studio operational view by discussing the system component interactions across the communications bus.

There are three main functional components in the Training Studio; the Vista virtual environment display, the VIVIDS instructional simulation, and the Steve Pedagogical agents. The VIVIDS component runs the simulation that controls the virtual world. Steve (one for each participant) monitors the state of the virtual world through messages it receives from VIVIDS. The Vista Viewer components (one for each participant) provide an interface between the virtual world and the human participants; they produce a 3D graphical rendering of the virtual world, and they detect interactions between participants and virtual objects.

Steve monitors the actions and field of view of the human participants through messages it receives from Vista, and Steve controls its own visual appearance by sending messages to the Vista.

Steve generates speech by sending messages to TrishTalk components. Each participant has his or her own TrishTalk, and each TrishTalk is an extended version of Entropic's TrueTalk text-to-

Figure 6. VET Training Studio architecture uses broadcast messages to maintain world state between components



speech program.

The Steve component uses another component, called RecAppl, to do speech recognition. RecAppl is a Java application that uses Entropic's speech recognition API to recognize student requests and speech acts for team members.

Another component, called the SoundServer, provides audio effects for objects. It acts as a service, responding to requests for audio effects. The SoundServer has been prototyped as a Java application.

All of the message passing among these components is accomplished via a communications bus using a message protocol called TScript. The overall system architecture, illustrated in Figure 6, is similar in some ways to other virtual environment architectures such as SIMNET (Calvin et al 1993) and Spline (Barrus et al 1996), in that it provides a common interface so that multiple applications can share access to a virtual environment over a network. One major difference is that multiple threads of interaction can take place at the same time, each using a different set of messages. For example, one set of messages is used to control the TrishTalk speech synthesis system, another set of messages is used to communicate changes to object locations in the virtual world, and yet another set of messages is used to communicate changes in the simulation state.

The communications bus allows an arbitrary collection of components to communicate. It is currently implemented on top of Sun's ToolTalk software. Components connect to the communications bus by sending a message, and they subsequently send messages announcing the types of messages they wish to receive. Components do not send messages directly to other components. Rather, all messages are sent to the communications bus, and the communications bus broadcasts each message only to those components that registered interest in it. This provides an important filtering mechanism; components only receive message traffic they can use. The approach also provides a more extensible architecture than direct component-to-component communication.

The VIVIDS component, which controls the behavior of objects in the virtual world, is a 3D extension of the RIDES 2D simulation authoring system (Munro et al 1993). In VIVIDS, as in RIDES, each object in the virtual world is assigned a set of attributes. Some attributes control the visual appearance of the object, while others control its behavior. The behavior of the objects is programmed by rules and constraints that propagate changes in one object to other objects. This object-oriented representation makes it easy for other programs, such as Steve, to monitor the state of the simulation.

A Vista Viewer component provides the interface between a human participant and the virtual world. Vista displays the objects in the virtual world in real time, and students can view the display either immersed, using position sensors and a head-mounted display, or in a flat-screen window. Multiple students, each with their own Vista Viewer, can connect to the same communications bus, and each will experience the same changes in the environment (albeit from their own viewpoint).

Vista acts much like an X server in X Windows. As each application starts in X Windows, it asks the server to create a window, as well as objects inside that window, and it expects the server to notify it of user events for those objects and windows. The server provides services for the display screen, and implicitly for the user. Vista acts more explicitly as a display server for the participant, but the idea is the same. Components such as VIVIDS and Steve ask that objects be created in the scene. Vista can build objects from graphical primitives, and it can also load them from various file formats, most prominently VRML 97. When the participant interacts with these objects, Vista sends event messages to the communications bus, and these messages are broadcast to other components that are interested. Participants can interact with objects using a variety of devices (e.g., Flock of Birds™ position sensors and the Virtex Cyberglove™), but the details of these devices are abstracted out by Vista in order to provide a generic set of interaction messages (e.g., selection of an object). Steve can send these same messages, and hence can interact with objects in all the ways that human participants can. Components such as Steve can also register interest in particular objects, in which case Vista will send messages when these objects come into or go out of a participant's field of view. Thus, Vista serves a dual role: it makes the virtual world real for students, and it informs the other components of the students' actions and field of view. The information provided by Vista is

crucial for allowing agents such as Steve to monitor the behavior of students.

TScript, which stands for Training Script, is the collection of messages that the components use to communicate. Object creation, modification, and deletion are controlled by TScript messages. Participant control for the purpose of instruction, such as changing a particular student's view or moving them along a path, are controlled by Tscript messages. Vista Viewers send notification of participant actions, their selection of objects, their movements in the world, and events they cause to happen indirectly all as Tscript messages. Each TScript message consists of a message name and arguments. The TScript protocol is extensible; the communications bus is not restricted to a fixed set of message types. Each component on the communications bus is free to define the set of TScript messages it can provide, and other components are free to register interest in the messages they need.

During a training session, much of the message traffic comes from VIVIDS. Since VIVIDS controls the behavior of objects in the virtual world, it must inform the other components when attributes of these objects change. Vista Viewers register interest in visual attributes, such as the location and color of objects. Steve agents register interest in the attributes they wish to monitor, primarily those that describe the state, rather than the appearance, of objects. For example, in the case of the High Pressure Air Compressor, this includes information such as the pressure in cylinders, and whether lights are on or off. Each component receives only those messages that are relevant to it. Moreover, VIVIDS only broadcasts changes in those attributes in which some component has registered interest. Such efficiency in message traffic is crucial for handling complex, dynamic worlds.

To illustrate how the components have been integrated using the communications bus, we close this section with a brief description of starting the system for one student, in the High Pressure Air Compressor domain, and give examples of how the components interact with TScript messages. Initially, a student starts up a Vista Viewer, which shows an empty 3D scene to the student. Then VIVIDS is started, and the HPAC course is selected. The 2D control panel for VIVIDS appears, as does a small window with an "Initialize" button. The student presses that button, and VIVIDS creates all the simulation objects and sends TScript messages to create corresponding 3D representations in Vista, either from graphic primitives or from pre-existing 3D model files. Then Steve is started, and its interface appears. Steve uses the communications bus to register interest in particular attributes of relevant objects, and it sends messages requesting the initial state of these attributes. VIVIDS responds with Tscript messages describing the state of those attributes.

Now the student is through with setup and starts the course using VIVIDS. They put on the virtual environment gear and select Start from the palette in their (virtual) left hand, immersed using Vista. VIVIDS progresses through the course, sending voice commands over the communications bus to the student's TrishTalk, which generates speech for the student. The student carries out requested actions, such as pressing buttons and opening valves. At each student action, Vista informs VIVIDS and Steve, and VIVIDS then can use the action to determine if the simulation state has changed. If it has changed, and Steve registered interest in the changes, they are broadcast to Steve.

At some point in the lesson, VIVIDS may send a message to request that Steve monitor the student performing a task. When the student needs assistance, he can touch a button on Steve's interface palette to ask a question, which causes Vista to send a message to Steve. Steve can answer the question by sending text to the student's TrishTalk, causing speech to be generated.

At other points in the lesson, VIVIDS may send a message requesting that Steve demonstrate a task, or the student may request a demonstration directly by touching a button on Steve's interface palette. Steve then carries out the task by sending messages to manipulate objects (these messages are handled by VIVIDS) and messages to move its own visual representation (i.e., body or hands) in the virtual world (these messages are handled by Vista). VIVIDS responds to Steve's actions by changing the state of the world and sending messages describing the changes.

All these interactions between components are carried out using TScript messages on the communications bus. There may be more than one student, each with their own Vista Viewer and TrishTalk, there may be more than one VIVIDS, each controlling the behavior of a different set of

objects, and there may be more than one Steve agent, each monitoring different students or demonstrating different tasks, but all the components maintain a consistent view of the virtual world via messages on the communications bus.

D.2 Technical View

In this section we relate the Training Studio technical view by describing the three main functional components, Vista (virtual environment interaction), VIVIDS (intelligent tutoring systems), and Steve (pedagogical agents). A technical architecture view describes a profile of a minimal set of time-phased standards and rules governing the implementation, arrangement, interaction and interdependence of system elements.

We have categorized the implementation areas addressed during the VET effort as operating system, software engineering services, user interface, data interchange, graphics, and networked update. We provide a brief itemization (see Table 3) before discussing the components.

For the operating system, we selected SGI's Irix operating system, since it drives the high-end graphics systems we needed to test our virtual environment concepts.

For software engineering services, we selected C++ as the high-speed graphics language of choice, and Java and Tcl/Tk as our 2D windows prototyping tools. The Profiler and RecAppl speech recognition components are built using Java, while the Steve interface was built using Tcl/Tk.

The 3D user interface standard we selected was VRML 97, which is an ISO standard. The sensors nodes in VRML describe constrained ranges of motion for 3D objects that are most often encountered. We worked to extend VRML for full 6DOF manipulation.

For data interchange, we developed the Tscript message protocol, that allows scene graph manipulations and queries between the Training Studio components. For interchange of 3D models, we used the Geometry nodes for VRML 97, and for interchanging behavior in a file-based manner, we used VRML 97 interpolators, route mechanisms, and script nodes which encapsulated a version of JavaScript. Detailed instructional behavior was interchanged using VIVIDs proprietary format.

For high-speed 3D rendering, we based our Vista component on SGI Performer, a COTS development tool, with select libraries in Inventor also being used.

For networked update of state across the Training Studio components, we used Tscript, and for update of 3D VMRL models, we used HTTP.

Service Area	*Service*	*Standard / Application*
Operating System	Execution Environment	SGI Irix 5.3+
Software Engineering Services	Prototyping Support	Tcl/Tk, Java
	Optimized Rendering	C++
User Interface	File-based 3D Interaction	VRML 97 Sensors
Data Interchange	Component Communication	TScript
	3D Models	VRML 97 Geometry
	Behavior	VRML 97 JavaScript VIVIDS
Speech	Recognition	Java, Entropic HAPI
	Generation	Tcl/Tk, Entropic TrueTalk
Audio	Sound Effects	Java
Graphics	3D Rendering	SGI Performer 2.0.2 Inventor 2.2
	2D Rendering	VIVIDs using X11
Networked Update	HTTP	Apache Client code for http services in Vista
	TScript	ToolTalk RPC

Table 3. Standards and COTS tools used for Training Studio components.

D.2.1 Virtual Environment Interaction

This section provides a technical view of the Vista component of the Training Studio, which is where

virtual environment interaction, between the student and the virtual environment, occurs. Virtual environment interaction consists of visual scene display to the participant, and a means for the participant to manipulate the visual scene.

Vista provides the visual context for the training process, and is the primary interface for the student.

Vista[1] provides stereo display and real-time interaction for 3D scenes where a student is immersed inside the 3D data - when Vista displays the engine room of a ship, they feel as though they are inside that engine room, and when they change where they are looking, the scene changes correspondingly. Vista, which was developed using the Silicon Graphics Performer toolkit (Rohlf 1994), also functions as a flat-screen display, where 3D scenes are shown through-the-window, for purposes of authoring and display on all SGI machines.

While immersed, the student can select items, such as selector switches, buttons, and valves and use them in part of a training simulation. The student can also move objects if this is required as part of their training task.

The capabilities of stereo immersive display and interaction with the scene objects are normally present in most virtual environment systems. Vista starts from this required base capability and extends it to support shared 3D scenes, monitoring the student's interaction, modification of the 3D scene, and modification of the student's view and interaction by external software.

D.2.1.1 External Modification & Control

Modification and control of the Vista 3D scene externally is accomplished by supporting the abstraction of named graphical objects. Vista understands networked commands for the creation of many types of 3D primitives, such as sphere, cubes, 3D text etc., and their placement in the scene graph. Vista supports loading CAD-derived models as graphic objects, and preserves named references associated with the geometry in the models. For instance, a bicycle could be loaded with references for the matrices controlling rotation of the pedals, wheels, and steering wheel. A named graphic object, such as bike1 could be loaded, and a second bike2. When the simulation controlling the bike wished to move the bike in a turning fashion, it would send TScript messages to any Vista displays in a session to rotate the handlebars, rotate the wheels, and rotate the pedals. All networked Vistas would show this resulting motion, and the second bike could be controlled independently at the same time, even though its model also has references with the same names.

In this manner, CAD models for ship items such as turbine shafts, doors, pistons, wheels, and valves can be tessellated and output as models suitable for loading in the polygon-based Vista display. After tessellation, the named references for articulation can be added. Instructional authors can use external tools to articulate and control these parts, knowing that the parts will be updated consistently in all Vista displays involved in a networked session. These named references also serve as identifiers during the student's interaction with parts. If the student select a given valve, all external training software interested in this selection action is immediately made aware of it. If a given simulation owns that valve, it simulates changes in the valve rotation accordingly. If a given tutoring software process finds that was not the correct valve for the task, it can react accordingly.

Most tutoring systems need to know what the student is doing as a first step in helping them if they do it wrong, or crediting them if they do it right. Vista supports monitoring student actions by external training software. General selection of objects is broadcast, as is the movement of objects by students, and the movement of their view and their hands. External components can register events with Vista to get more detailed information, such as which objects are currently visible to the student, which objects have entered a given space, and which segments have intersected with object geometry.

Since Vista moderates a student participant's interaction in a virtual environment during the course of training, at startup it provides for configuration of the student's parameters; which VE devices they are using, the distance between their eyes which is used for stereo vision, their height, student name, etc. Vista also provides other Training Studio components with references

[1] The Vista system initial development was sponsored by the United States Air Force Contract No. F41624-93-C-5000.

to the student's view and hands, so that interfaces or explanatory objects can be attached to their view or hands. The Training Studio component Steve uses this capability to attach an interface palette to a given student's left hand. After that point, the Steve software does not have to devote effort to maintaining that palette, Vista does.

The Training Studio provides the instructor with a variety of tools to control student movement and attention as required. Instructors can designate that students follow a specific path or be moved instantaneously to a particular location. Student viewpoint can be explicitly directed or left uncontrolled while traveling a path or remaining at one location. Other tools for directing attention include object highlighting and color cues. Also, a bounding area can be designated around the object so that student intersection with this area, triggers an instructional sequence. This ensures that the instruction plays when a student is in position to observe it.

All the components of the Training Studio interact to provide the instructional lessons and environment. Because the participant is free to move within the environment, it is difficult to predict the exact location and orientation during run time. The simulation (VIVIDS) and agent (Steve) components can query the state of particular objects or participants by sending a TScript message via the communication bus. The Vista environment replies to the query with the appropriate data.

D.2.1.2 VRML Capability

Vista supports networked training applications, both in terms of TScript messages sent over the local communications bus, and in terms of VRML on the Internet. Vista is capable of using the http Internet protocol to fetch files, and exercises caching of retrieved files to increase apparent efficiency. The primary Internet file format supported by Vista is the ISO standard Virtual Reality Modeling Language (VRML 97) specification.

The Virtual Reality Modeling Language (VRML) is a language for describing 3D scenes and the behavior of objects in those scenes which are delivered via the Internet. It is a platform independent file format and is quickly becoming the standard for 3D graphics interchange on the Internet. VRML, which is derived from Silicon Graphic's Open Inventor file format, is designed to work well over low-bandwidth connections. Translators exist which convert most popular 3D file formats to VRML, and many modeling programs have built support for directly editing and outputting VRML model files. This has made VRML popular and many models developed on various platforms are available for re-use.

The VRML 1.0 specification evolved on the WWW-VRML mailing list. We have been active participants on this mailing list and have made contributions towards the VRML 1.0 and VRML 97 specifications since April, 1994. VRML 97 is concerned with dynamic virtual worlds, in contrast to VRML 1.0 which supports static 3D scenes with WWW links embedded. VRML 97 is a further step towards the goal of describing moving, multi-participant interactive virtual worlds linked via the Internet.

The VRML format consists of various nodes arranged into a tree graph, and connections between those nodes to achieve simulation. There are many types of VRML nodes, but we discuss those most relevant to training here, as well as how VMRL nodes communicate in Vista. VRML 97 supports 3D geometry display (Geometry Nodes) event generation (via sensor nodes) and event-passing between nodes in the scene, and interpolated object changes (Interpolator nodes). VRML 97 nodes can have behavior attached to them via scripting languages (e.g. Java, JavaScript, in Scripting Nodes), and VRML 97 provides a set of powerful tools for exchanging and using virtual worlds.

The Anchor and Inline nodes in VRML are the primary way in which VRML uses the Internet. Whenever an Anchor is selected in Vista, a new 3D scene is downloaded from a site on the Internet, to replace the current scene. Inline nodes are more complex, but they allow an instructional developer to assemble a composite scene from many component scenes, which may distributed at many sites over the Internet.

To optimize VRML downloads for large models, in-lined VRML files are only transferred when the user is near them, and when the user is further away, less complex geometry is displayed. This is accomplished using Level of Detail (LOD) nodes with Inlines as children. This LOD node delay loading of inlined VRML files until necessary. This is very useful when the inlined files are large (e.g., the model of a ship may be

made up of several inlined files for decks, pipes, and machinery which are loaded only if the user is close to that part of the ship. In this way waiting for VRML models to arrive over the network is minimized).

Sensor nodes in VRML 97 support user interaction. Events generated by sensors can be routed to other nodes via routes. The VRML 97 sensor nodes (Proximity sensor, Touch sensor, Cylinder sensor, Plane sensor, Sphere sensor, and Visibility sensor) generate events based on user actions such as a mouse clicks or navigating close to a particular object. Time sensor nodes generate events at regular intervals. Sensors provide a very useful tool for developing training scenarios, e.g., a cylinder sensor attached to a gauge on the HPAC would cause the gauge to respond to user action. Similarly, a time sensor could be used to start an animation in a training scenario at a particular time and the proximity sensor could be used to initiate action when the user enters a particular region in the training scenario.

Interpolators are used to update values for animation and illustration independent of network delays. Given an input value an interpolator evaluates a linear function to arrive at a value from a set of values. There are different type of interpolator nodes in VRML 97 depending on the type of value being interpolated. Vista supports the Color Interpolator, Orientation Interpolator, Position Interpolator and Scalar Interpolator. Interpolators provide a useful tool for creating training scenarios; e.g., a time sensor attached to a Position Interpolator could be used to move an engine cylinder over time.

A ROUTE in VRML 97 is a connection between a node generating an event and the node receiving the event. A ROUTE class has been developed to handle routes between VRML 97 nodes in the Vista viewer. The significance of routes lies in the fact they propagate user generated events in a scene graph thereby making it dynamic. Routes allow events generated by nodes to be connected together, forming a path.

In order to handle events between VRML 97 nodes in Vista an Events class was developed. Instances of this class are contained in VRML 97 nodes. This class is used for event-passing and creating routes between VRML 97 nodes in Vista. Events generated by one node can be wired to other nodes via routes thereby making the VRML scene dynamic. The event-processing model of VRML 97 provides a very effective tool for carrying out efficient simulation as required for training.

The primary emphasis for Vista has been connectivity to other software systems, and flexibility in configuring virtual environment scenes. Using TScript messages, or VRML 97, or both, many forms of virtual environment display and interaction are possible, including live information display onboard ship. Live information display, where equipment sensors output is processed by a software program, and then sent to Vista, is possible. Sensor readings could be displayed as colored bars near the 3D models of the equipment they collect readings for. Critical pieces of equipment located across several decks or multiple platforms could be displayed with this sensor data to view systems readouts in a way not normally possible while onboard ship. Such sensor readings and associated Vista display could then be used in training students, in this case from live recorded data.

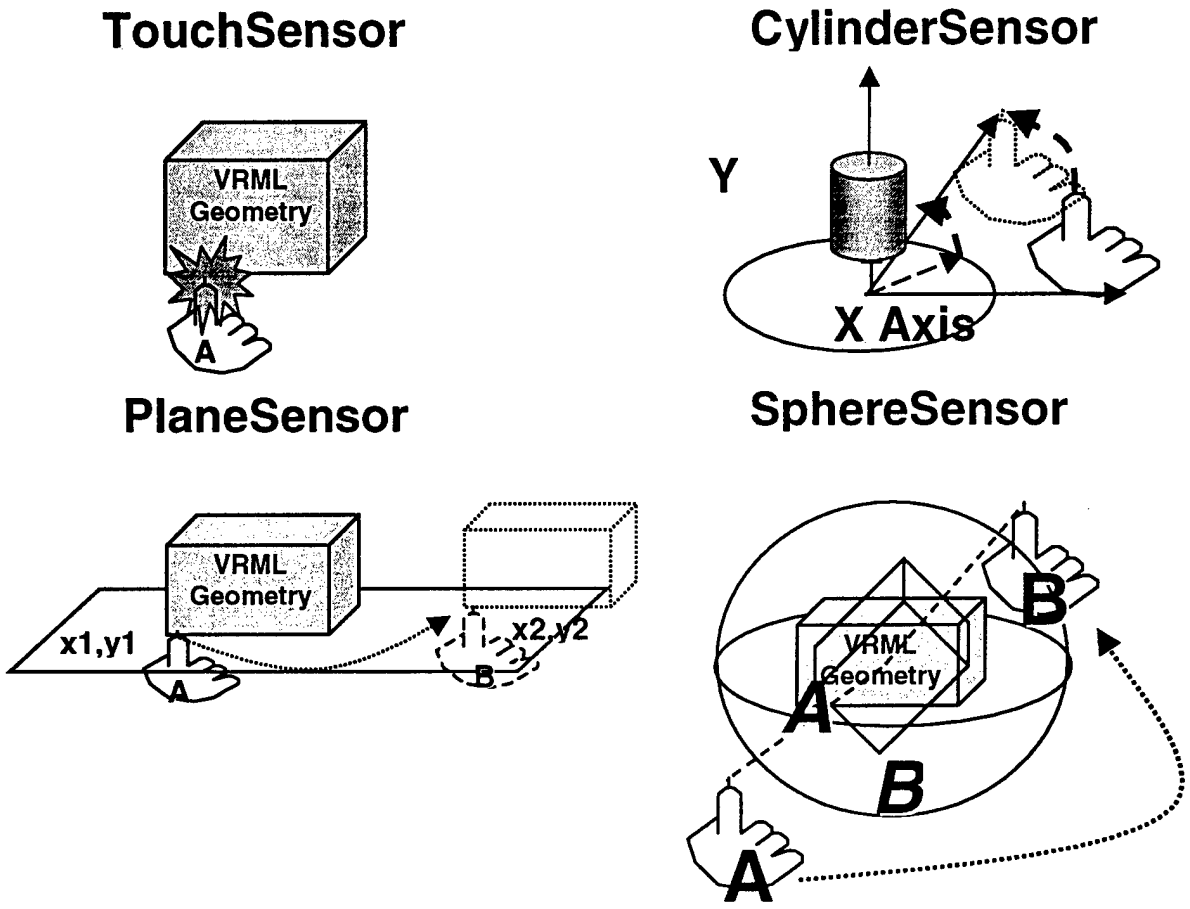
D.2.1.3 Immersed VRML Manipulation

This section describes Vista's capability for immersed manipulation of conventional VRML 97 sensors, as well as extensions to VRML for full six-degree of freedom manipulations (6DOF). Also described is the capability to manipulate all sensors with two hands immersed.

VRML 97 provides mechanisms for interacting with moving, dynamic scenes. For VRML 97, human-computer interaction for the mouse has been fairly well defined and tested. However, a person carefully reading through the standard will note the mention of 3D pointing devices, such as a wand, in sections describing Sensors. Here we discuss an immersed interaction approach for VRML 97 scenes, and an implementation. We also provide refinements or cautions for those VRML 97 node types that can be interpreted differently in an immersed setting.

The Vista capability for immersed manipulation of VRML sensors can be used in an ego-centric manner, or an exo-centric manner. The egocentric manner is direct manipulation by intersecting (touching) sensors, and the exo-centric manner is projected manipulation, where the user pinches their fingers visually over the

Figure 7. conventional VRML 97 sensors can be directly manipulated while immersed



object, at distance, which is similar to a mouse. A useful taxonomy of usability characteristics for virtual environments is (Gabbard 1997). For VRML interaction, we follow a number of the recommendations in this report, notably a uniform object selection approach, and a two-handed manipulation approach that recognizes the dominance difference between hands (see Figure 7, Figure 8, Figure 10).

Before discussing the interaction differences between VRML 97 immersed and flat-screen, it is useful to explicitly indicate the similarities. For the most part, nodes included in the VRML 97 categories of Leaf Nodes, Geometry, Appearance, or Geometric Properties are no different immersed from flat-screen. All the Group nodes have a direct interpretation immersed, including the Billboard Node. Finally, the Interpolators do have the same interpretation flat-screen or immersed. The primary difference is in the use of Sensors and view modifying nodes.

When using a head-mount display to view a scene, the user can very quickly rotate his head in different directions to view different parts of the scene, while independently moving his hands for interaction. Generally, the user has greater flexibility when interacting with 3D objects immersed than using a flat screen and mouse. This flexibility comes with some problems that don't occur in flat-screen mode; the user can select objects not in their view, events may be uselessly generated as they quickly change view, programmed view transitions must be handled carefully, and an absence of force feedback may result in unnatural interactions.

To deal with out-of-sight selection problems immersed, our VRML 97 implementation does not

generate any events if the intersection point with VRML geometry is outside the field of view.

To interact with VRML 97 scenes the user must be able to use interaction devices to touch objects and trigger sensors. In flat-screen browsers, the user interacts with objects in the scene using the mouse only if an object is visible. In an immersive environment, depending on the interaction approach, the user may need to navigate close to the object even if it is already visible. So for immersed users, there may be additional steps involved in interacting with objects. In the case of direct interaction using a glove, the user would have to get very close. The advantage for direct interaction is the possibility of directly manipulating objects, rather than having the user make a cognitive mapping from a moving mouse to the actual motions of a 3D object.

Another interaction difference occurs when viewing behavior is triggered as a result of user action. In flat-screen mode, interacting with parts of the scene often doesn't have to need to reposition to see the behavior generated by this action. This is because the user is often at a distance from the scene when interacting with it. In an immersive environment the user may need to be close to the object he in order to interact with it. After interacting with the object the user might have to pull back in order to observe the resulting behavior. This required when the size of the object is large because when the user is close the object occupies a large area of his field of view. We observed this to be a problem when we used the glove to interact with a robot in a VRML 97 scene.

An immersed user can directly rotate and move objects in all six degrees of freedom. Because the user has more freedom of motion in an immersive environment than in flat-screen mode, it is harder to restrict the user's interaction with sensors to a plane, cylinder or sphere. The immersive VRML 97 implementation should use alternate strategies to deal with any disadvantages of this freedom in a 3D setting.

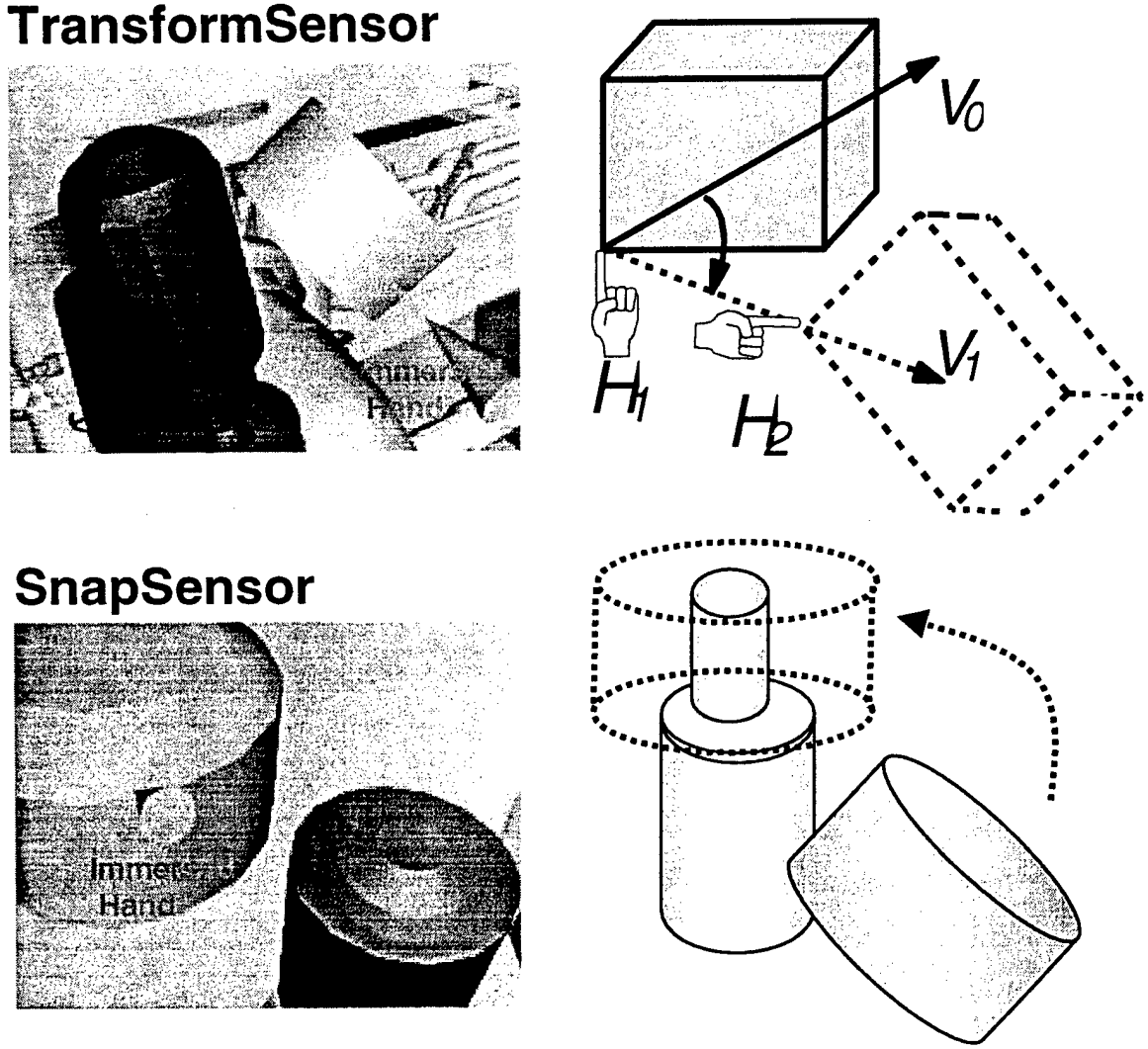
Without force feedback, it is too easy to put your hand or other manipulation device through an object, since there is no force to limit the hand to the surface of an object. Therefore directly placing your hand and applying pressure to rotate or move an object doesn't work well. However, an interaction approach that allows the user to manipulate objects, as near to touching the object as one would in reality, is still desirable. We have arrived at an approach that allows nearly direct manipulation, but still adheres to the VRML 97 interaction principles for a mouse that can make interaction at a distance easier.

Viewpoint nodes are used to provide different views of the scene. A ProximitySensor can be used to trigger a Viewpoint node and so if a user is in a particular region then his view is attached to a Viewpoint node. If this viewpoint moves then the user's view also moves with it. In flat-screen mode a sudden change in viewpoint is okay because the user is not immersed. In an immersive environment a sudden change in immersed viewpoint can adversely affect the user; they can become disoriented. We propose that a change in immersed viewpoint should be gradual, similar to the effect achieved through orientation and position interpolators.

ProximitySensors can be very useful for reducing the complexity of the scene when immersed. In VRML 97, in addition to the geometry being drawn, events are processed in response to user action. Sensors and script nodes which add behavior to the scene can be culled (switched off) depending on where the user is in the scene. Events for behaviors not in the user's proximity should not be processed. This is true for flat-screen browsers as well, but is more critical immersed.

In an immersive environment the user has more freedom to look around, and expects to be able to look around. The VisibilitySensor is very useful for finding out where the user is looking. This can be of great help in training scenarios where the trainee(s) might not be looking where they are supposed to, since VisibilitySensors can be used to determine if the user is looking at the desired object. The other use of a VisibilitySensor is to determine if some behavior must be started. Since in an immersed environment the user has greater flexibility, the user could potentially initiate behavior unintentionally as they sweep their view to look in another direction. For most cases, events should be passed and behavior of objects displayed only if the objects are visible. Also, the behavior should start only if the user looks at the object for a minimum time, such as a default of five seconds. This could help prevent the display frame rate from dropping because of unnecessary event generation.

Figure 8. TransformSensor and SnapSensor allow assembly of VRML objects using full free-form motion (6DOF)



We have extended the VRML standard to allow free-form manipulation of objects while immersed. Our driving goal was immersed training for equipment operations and maintenance, and to this end we developed a sensor that allows 6DOF manipulation, a cooperating sensor that allows snapping objects into place as part of an assembly, and a two-handed manipulation approach for these sensors.

We discuss our implementation of TransformSensors and SnapSensors, for both single and two-handed manipulation of objects. Free-form manipulation of objects is a necessary prerequisite to our work in applying virtual environments to training. We are supporting operations and maintenance training on CAD-derived shipboard equipment, where it is a common task to pull objects out, assemble them, and snap or plug them into assemblies.

A TransformSensor (see Figure 8) is used to designate an object as being moveable in all six degrees of freedom; i.e., by changing translation and rotation concurrently, such as is possible with a 6DOF position sensor (Ascension, Logitech, Polhemus, etc.). The changed rotation and translation can be routed to a Transform node, Script node, etc.

The SnapSensor (see Figure 8) is used to designate certain locations in the scene graph as snap locations. By using a SnapSensor, the content author can specify that a given type of object will fit into that location. The SnapSensor holds the position and orientation of the object when it snaps. This is useful for designating a location where a nut will fit into a bolt, a shaft will fit into a casing, etc. Most importantly, snapping allows the author to overcome imprecision in movements that is common in immersed systems that have no physical feedback for hard surfaces. Our technique for snapping is based on range checks.

To support equipment maintenance and operations training, it is necessary to let people tear apart objects and put them back together, to replace parts or open them to inspect them. For realistic training and effective evaluation of skills, a level of freedom during performance is often required; i.e., a single "correct" path cannot be pre-defined or, therefore, pre-authored. Multiple solution paths can exist for reasons relating to both procedures and the object itself. Procedural differences are common to real world behavior and can be due to a trainee's reordering of sub-tasks that are independent, and not strictly hierarchical. Issues involving free manipulation also arise when authoring the behaviors of models used for equipment maintenance and operations training. Often assembly of equipment involves several objects that may or may not be functionally interchangeable, but are physically similar.

The primary techniques we selected as critical to support real-world manipulation during training include 6DOF manipulation, two-handed manipulations, and snap locations for object placement. Allowing for 6DOF manipulation is particularly important when providing realistic interactions within the environment. More restrictive manipulations would probably suffice for operations at consoles or panels, but maintenance often requires more complex interactions. Support for combined translation and rotation of objects is essential for tasks such as part replacement or component assembly.

Provision for two-handed manipulations is important if the experience is to extend to real world interactions. For example, to remove a large panel or other piece of equipment, the object must sometimes be pulled straight out or at an angle, which requires two hands on a single object. For this type of manipulation, each hand plays a role in positioning and orienting the object.

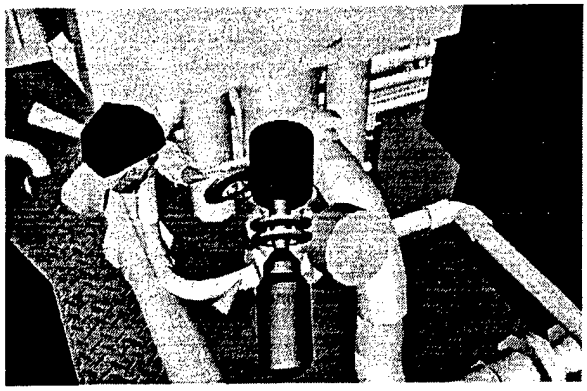


Figure 9. Immersed two-handed manipulation of pump assembly, small green arrows represent hands

Two-handed manipulation is also needed for simultaneous manipulation of multiple objects. In the example of disassembling an oil pump (See Figure 9), three rings are removed from a shaft previously removed from the pump. The shaft must be held with one hand while the other removes the rings. Manipulation techniques must be available for both hands, each controlling a different object.

Two-handed, immersed manipulation of VRML TransformSensors is accomplished in the Vista Training Studio component using the non-dominant hand for gross translation of the object, and the dominant hand for fine orientation of the object, similar to the two-handed approach by (Cutler 97) and (Guiard 94) (see Figure 10). The steps below describe the two-handed approach:

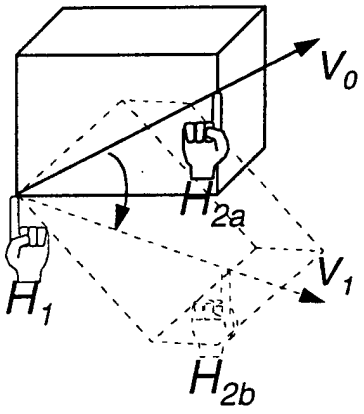


Figure 10. Two-handed manipulation, non-dominant hand translates, dominant hand orients

1. The first hand that does the pick operation determines the object to be manipulated (Roll from the first hand is ignored).
2. Translation is determined by the first hand to select the object.
3. The second hand intersects the same object and the segment between first and second intersection points on the object defines a vector, V_0. Its direction is from the first intersection point to the second.
4. Either one or both hands are moved and the segment between the two endpoints of the two hands determines Vector V_1. No further intersections with the object are required.
5. The object is rotated so that initial vector is aligned to the new vector as the two hands change locations.

Another issue involves connecting objects within the immersed environment. This is essential for equipment operations since the assembly of components requires joining objects, for example, screwing in a bolt, joining two pipes, or inserting alignment screws. These are tasks that require precise object alignment, which is difficult in the immersed environment. Requiring such precision in an environment that lacks tactile feedback might affect the training process since the focus shifts from the training content to the training environment. Instead, we simplify matters so that if the trainee places the appropriate type of object in a location that is authored to accept the object, then the object will automatically snap into place.

D.2.1.4 Concurrent Interaction

Immersed virtual environments provide greater freedom of movement and views than more traditional forms of computer-based learning. This works well for allowing students to experiment and familiarize themselves with their surroundings, however, there are times that an instructor must ensure that interaction with an object is perceived by the student or team, and must perceive with fluid situations created by a team.

Vista provides a number of services that are used by the Steve and VIVIDS simulations to support human/scene interactions. There are services to determine world coordinates and bounding areas of scene objects, and modify scene objects (color, transforms, visibility, etc.). There are services to allow students to interact with objects (Stiles 1997, 1998), and for the changed object state to be updated for all the team. Services also exist to determine if an object is visible to a student, and to control the student's view for instruction.

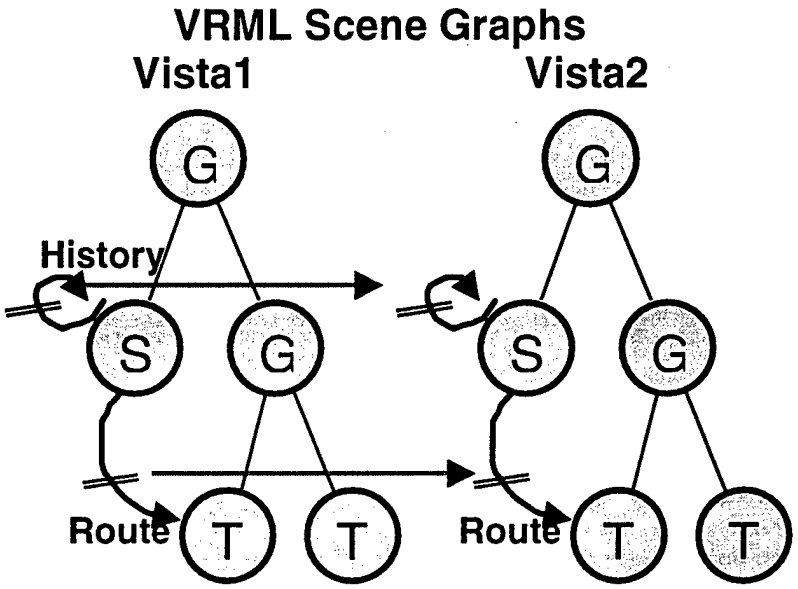
Events can occur in a virtual environment which are not perceived by student's with a free-ranging view of the 3D scene. This problem is compounded when multiple participants must observe an event since multiple views and locations must somehow be transitioned to the desired view.

The Vista Viewer provides the instructor with a variety of tools to control student movement and view. The instructor can transition the student on a set path or can instantaneously move the student to a new location. The latter works well for situations in which the participant is fully aware of the shift, for example, has pressed a location on a map. However, merely snapping all participants to a desired viewpoint to view a demonstration, for example, could be disconcerting and disorienting. Vista implements viewpoint transitions by moving the user from the initial destination to the desired orientation by moving slowly initially, then speeding up towards the end. The movement begins slowly to allow the participant to become better oriented and establish a spatial relationship between the two views. Steve can wait until all students are in position to view the event before actually triggering the event. This is done by the use of proximity sensors located on the object that will be viewed. The proximity sensors communicate, by identity, when a user enters or exits the defined area.

During normal operation each participant has one Vista Viewer assigned to him. The introduction of multiple viewers requires coordination of each viewer to ensure scene consistency between participants. VRML sensors, for example, allow the environment to respond spontaneously and naturally to an individual's action. Respective sensors in the other displays, however, must be triggered to ensure that all participants share the same view. Likewise, it is not enough to provide a representation of other team members for individual displays; the actions of team members (and results from those actions) must also be broadcast and displayed. Views of participants must be tracked and coordinated to ensure that specific events are viewable by all required participants.

We currently update most of our training simulation state visually by means of TScript messages sent from the training simulation to all Vista Viewers. We also automatically route VRML sensor values generated by a student participant in one Vista to other Vista displays (see Figure 11). The VRML routing of a sensor event goes across the network, and the event is routed as though it had occurred internally to the receiving Vista Viewer. This is a necessary capability to share visual state as actions occur. Currently the underlying training simulation is always in synch with the Vista Viewer making changes to the state, but the simulation does not update all other Vista Viewers to reflect the change. We could keep Vista displays synchronized in this manner, but it involves an additional message for every event and wouldn't scale up to larger simulations well.

Figure 11. VRML scene graphs are updated across network by broadcasting sensor changes along routes, and no modification of VRML file is needed



To support individual remediation and instruction, our approach also allows for private (per-participant, temporary) changes to the simulation/scene to an individual display for ad hoc lessons. Once the remediation is complete, the individual's scene must update to allow him/her to rejoin the scene.

D.2.1.5 Authoring with Commercial Tools

The full spectrum of 3D modeling is quite complex, and often authors building 3D scenes need many different tools to get the effect they desire, or to work with the data they are given. Early in the VET program, we embraced the use of ISO standards for 3D geometry commercial-off-the-shelf (COTS) authoring technology for 3D scenes. The result was that we built Vista capabilities to load and interact with VRML files in support of a commercial authoring capability.

Many commercial tools are available to edit 3D models and export as VRML scenes that can be used immersed in the Vista Training Studio component. Commercial VRML authoring tools include CosmoWorlds, VRCreator, Sony Community Place Conductor, Caligari TrueSpace, and Studio3D. Commercial CAD vendors such as Intergraph, Parametric Technologies ProEngineer and SDRC Ideas provide VRML export capability.

The typical authoring process begins by obtaining 3D models for the training domain, and converting these to VRML. Once the conversion to VRML is done, the scene graph structure of the VRML scene should be modified to support later manipulation and interaction. For instance, if an engine assembly object has both the engine and valves as one complex set of geometry, and the valves must be manipulated for instruction, the valve geometry must be separated from the complex engine geometry, so that VRML sensor manipulations can turn the valves without turning the entire engine.

Then the VRML models detail is reduced to acceptable polygon counts, colors, materials, and textures are changed to more closely approximate the real equipment, and equipment items that will be manipulated are named for reference in Vista by the other Training Studio components, such as VIVIDs.

Then interaction with the VRML objects is authored, using VRML sensor nodes. Doors, dials, thrust controls, valves, etc. all can be

manipulated by CylinderSensor nodes. Loose items of equipment, and those that are assembled, are modeled using TransformSensors and SnapSensors. Sliders and plane-constrained objects are authored with PlaneSensors, and free-rotating joints are authored with SphereSensors.

Once constrained manipulations have been specified with VRML sensors, the referenced nodes in the VRML scene graph geometry are made available to VIVIDS authors using the Profiler Training Studio component. The VIVIDS author loads the VRML scene into VISTA in one window, uses the mouse to select objects in the VRML scene, and sees their reference name in the Profiler. Using this reference name, rules and object behavior can be edited in VIVIDS, and the effect can be tested out in Vista, all in the same session.

D.2.2 Simulation-based Training

This section describes the VIVIDS component of the Training Studio. VIVIDS is used to author and deliver Training Studio equipment simulations and related instruction.

D.2.2.1 Authoring 2D/3D Equipment Simulations with VIVIDS

Intelligent tutoring systems (ITSs) often include interactive graphical simulations. For many types of tutoring, the use of an interactive graphical simulation helps to assure that what students learn is relevant to actual tasks that they must learn to perform, in a way that a primarily textual or static graphic approach to learning interactions cannot. Interactive simulations can help to ensure that performance skills—as opposed to mere test-taking skills—are acquired as a result of tutoring. To date, most research projects on intelligent tutoring systems that have incorporated simulations have relied on low level tools (i.e., programming languages) to develop both the ITSs and the simulations. Reliance on such low-level development techniques naturally can make simulation-centered tutoring extremely expensive. It can also make it very difficult to determine what features of a particular tutor are responsible for its efficacy. The of low-level development using programming languages can overwhelm the effects of general principles that are followed in a particular tutor. An authoring system, by providing easily edited and modifiable tutorials, can make it possible for developers to experiment with different high-level approaches to tutoring in a given domain.

VIVIDS is a descendant of the RIDES application for interactively authoring graphical simulations and simulation-centered tutorials. VIVIDS (and a version of VIVIDS that lacks authoring features, called *sVivids*—for *Student VIVIDS*) delivers simulation-centered tutoring to students. Because the simulation authoring system is designed to support tutorials, many types of instruction can be very rapidly authored, and many high quality instructional interactions are generated automatically. VIVIDS, unlike RIDES, can be used to develop and present simulations and tutorials in the context of the Vista Virtual Environment and in collaboration with the Steve pedagogical agent.

D.2.2.1.1 History

The field of simulation in intelligent tutoring systems (ITS) research is a large and rapidly growing one. The field of simulation-based tutor *authoring systems*, however, is a very much smaller one. In this section, we briefly discuss several authoring systems for the development of simulations for learning.

STEAMER (Williams, Hollan, and Stevens, 1981; Hollan, Hutchins, and Weitzman, 1984) provided a direct manipulation simulation for students learning about a steam propulsion plant. The STEAMER project is an important spiritual ancestor of VIVIDS. It offered a discovery world for students and a demonstration platform for instructors, but it did not provide authoring tools for the development and delivery of instruction to the learner. Simulations had to be written as conventional computer programs.

Forbus (1984) developed an extension of STEAMER called the *feedback minilab*, which could be used to produce interactive graphical simulations without first writing separate simulation programs. This early authoring system provided a set of predefined components (such as valves and switches). Composing a device of these components determined the behavior of the simulated system as a whole.

IMTS (Towne and Munro, 1988, 1992) provided tools for authoring interactive graphical simulations of electrical and hydraulic systems. A library of graphic behaving objects could be composed, but the external effects of these

objects had to be of either electrical, mechanical, or hydraulic type. IMTS supported troubleshooting assistance by a generic expert called Profile (Towne, 1984), but it could not be used to develop or deliver other kinds of instruction.

An early approach to direct manipulation instructional authoring was Dominie (Spensley and Elsom-Cook, 1989). That system, however, did not support the independent specification of object behaviors; the specification of simulation effects was confounded with the specification of instruction.

RAPIDS (Towne and Munro, 1991) and RAPIDS II (Coller, Pizzini, Wogulis, Munro, and Towne, 1991) were descendants of IMTS that supported direct manipulation authoring of instructional content in the context of graphical simulations. These systems provided a more constrained simulation authoring system than is found in VIVIDS, and they did not offer authors low level control over instructional presentations.

RIDES and VIVIDS provide much more robust simulation authoring tools and instructional editing facilities than were to be found in RAPIDS and RAPIDS II. The VIVIDS system has some features in common with the SMISLE system (de Jong, van Joolingen, Scott, deHoog, Lapied, and Valent, 1994; Van Joolingen and De Jong, 1996) developed by a consortium of European academic and industrial research groups, but is less restrictive about how simulations can be structured. SMISLE authors must separately specify an inner, 'real' level of behavior and one or more surface depictions of the behaving system. Similar effects can be achieved using VIVIDS, but they are not required. The SMISLE system also contains facilities for supporting student hypothesis formation, but lacks the unconstrained simulation authoring and instruction authoring capabilities of VIVIDS.

D.2.2.1.2 VIVIDS Adaptations for Collaboration with Vista

VIVIDS was modified in the course of this project to respond to user actions in Vista and to request that Vista make appearance changes based on computed simulation effects. Actions taken by students in the Vista VE are reported to VIVIDS. The VIVIDS simulation associates these actions with corresponding actions in a 2D simulation and uses the 2D simulation to compute the effects of such actions (as well as any effects due to the passage of time, etc.). Whenever one of the authored 2D simulation objects undergoes a change in an appearance attribute, it broadcasts to Vista a directive to make a corresponding graphical change.

Several VIVIDS simulation objects have been built that can collaborate with corresponding VE model objects. Figure 12 shows the VIVIDS library interface being used to access a behaving rotary knob object. An instance of a graphical model object under the control of such a simulation object appears in the Vista scene shown behind the VIVIDS library dialog.

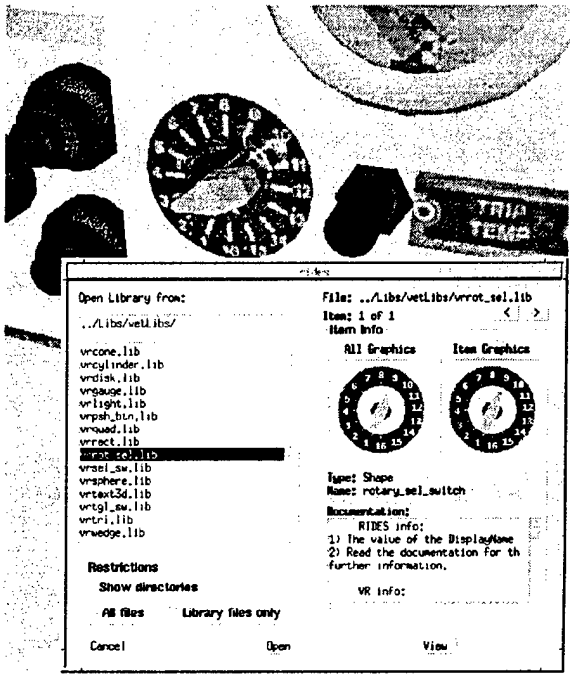


Figure 12 Simulation objects from VIVIDS library allow re-use of many common instructional items

D.2.2.1.3 Simulation Development in Parallel

Both 3D graphical model development and simulation behavior authoring can be time-consuming tasks, especially when they are carried out for a large and complex training simulation such as the Gas Turbine Engine (GTE) control systems. We have found it productive to engage in these two aspects of simulation development in parallel. While the Lockheed team developed GTE VRML graphical models using CosmoWorlds, a VIVIDS simulation behavior model was developed at Behavioral Technology

Labs. The major coordination that was required was that there be agreement about the names of the model objects that would exhibit behaviors by permitting student manipulation. The simulation author was able to quickly sketch 2D versions of simulation objects and focus on writing the rules for controlling their interactive behavior. The name of the 3D model graphic was entered into a VIVIDS simulation object data field so that the object could know where to address its graphical change directives.

In this way, entire simulations can be developed and their behavior largely debugged by a person or team at one site, while at the same time another is building the graphical models. It is also possible to build knowledge units (described below) and even preliminary structured lessons before the 3D graphical model has been integrated with the behavioral simulation.

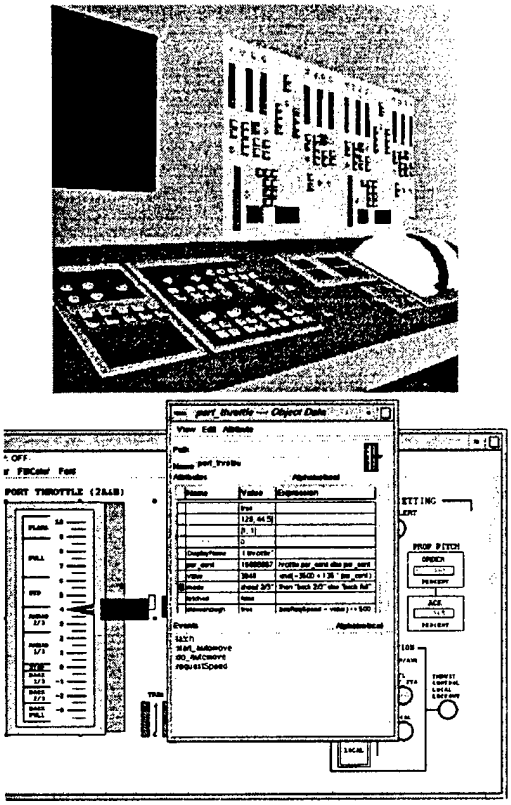


Figure 13. Independently Developed 3D Model and Behavioral Simulation are part of concurrent authoring approach

The 2D simulation that was roughed out independently of the 3D model included 31 simulation scenes--separate windows that graphically represent portions of the GTE control systems. One such scene is shown in the lower part of Figure 13. The author has selected a simulation object that represents a throttle control (note the black selection rectangles around the object) and has opened an *object data view*, which is used to enter the behavioral rules that determine how this object responds to student manipulations, and how it is affected by other objects in the simulation. The object data view is also used to enter the name of the 3D model node that corresponds to this behaving 2D object, along with other data about how the object can control the corresponding 3D graphics. In the 3D model shown above in the figure, this throttle object is more realistically rendered at the right side of the view.

D.2.2.2 VE Patterned Exercises and Custom Lessons

The VIVIDS system supports the rapid authoring of structured lessons for delivery in both 2D and 3D viewing environments. The lessons are authored in the context of the 2D simulation, which lets the author focus on pedagogical presentations and what student actions to require without having to deal with 3D navigation issues.

Two types of techniques for building structured lessons are provided. *Patterned Exercises* are lessons that are based on fifteen lesson templates that are built into VIVIDS.

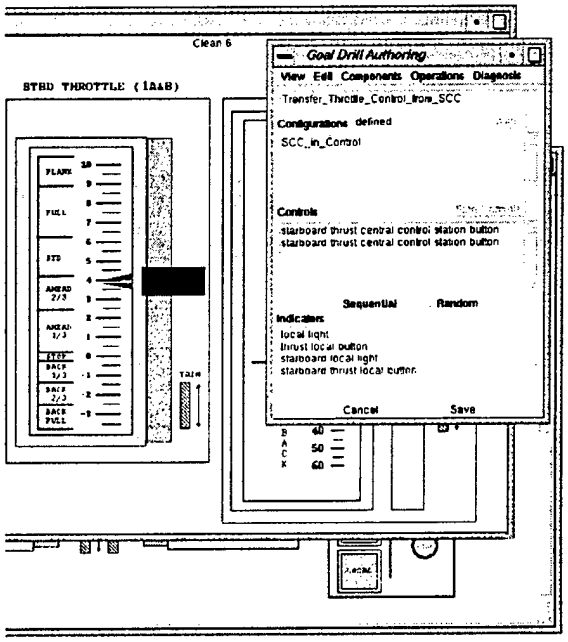


Figure 14. Building a Procedure Exercise with a Patterned Exercise Editor

In Figure 14 above, a type of lesson on how to carry out a procedure so as to achieve a goal is being authored. The instructor simply carries out a sequence of steps that will bring about the goal. As each action is taken, the name of the manipulated object is added to the list of actions shown in the editor. Once the instructor has achieved the goal, he or she indicates that no more actions are required and then points to those objects that indicate that the goal of the procedure has been attained.

During training, a student is asked to carry out the required procedure. VIVIDS evaluates the student's progress in terms of the indicators that the author pointed out. If the student is unable to do so, VIVIDS guides the student to carry out the sequence of actions that the instructor authored using the patterned exercise editor.

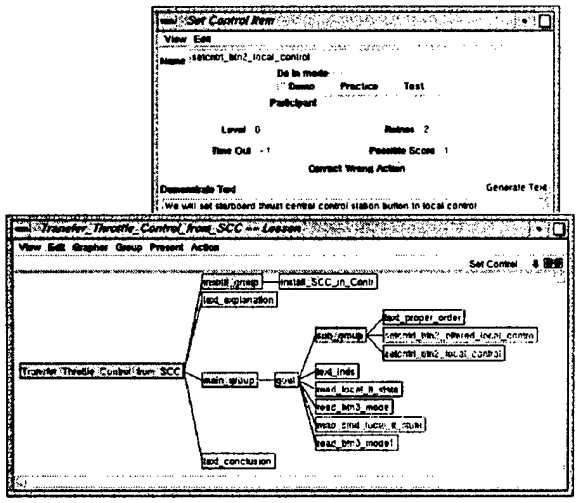


Figure 15. VIVIDS supports custom lesson development

It is also possible to build novel types of lessons that are not based on instructional templates. This is done using a *custom lesson editor*, shown in Figure 15. In this editor, an instructor can create new presentations, can specify questions to ask the student, and can author sequences of required student actions by carrying them out in the simulation. The custom lesson editor can also be used to edit lessons that were originally built using the patterned exercise editor. The lesson shown in Figure 15 is the one that was generated by the goal/procedure authoring process described for Figure 14, above.

No matter which approach is used to develop a structured lesson, the elementary steps of the lesson automatically handles a good deal of student interaction without requiring explicit author decisions. For example, objects that students are to select or to manipulate can be automatically highlighted by the VIVIDS lesson to make them visually salient as part of the instructional remediation process.

Structured lessons can be presented to students based on an authored course structure in collaboration with individual student models. The objectives of a course are associated with lessons that are designed to achieve those objectives. Some objectives require others as prerequisites. As students attain objectives, they become eligible for lessons designed to realize more advanced learning objectives.

D.2.2.3 Opportunistic Instruction

In addition to presenting a sequence of lessons to achieve the objectives of a course, authors sometimes find it useful to specify that certain lessons should be presented on specific occasions. For example, if a safety principle is violated by a student carrying out a simulated procedure, the author may want VIVIDS to interrupt with a brief lesson that drills the student on the violated safety requirements. This type of instruction is called *opportunistic*. Authors develop brief lessons using one of the standard authoring techniques. Then a trigger condition is authored that specifies when the lesson should be presented. It is possible to opportunistically present only a part of an authored lesson. When the opportunistic instruction is finished, the interrupted lesson resumes.

D.2.2.4 Accessing Authored Knowledge

VIVIDS provides an authoring mechanism called the *knowledge unit editor* for entering textual discussions about topics that can be associated with simulation objects. Authors can define both topic names and the content of the discussions for each topic. It is also possible to associate one or more structured lessons with a knowledge unit, and to enter key word indices that can be used to search among knowledge units. Figure 16 shows a knowledge unit editor being employed to enter a discussion about the Function of a simulated control switch in the simulation of the gas turbine engine control system.

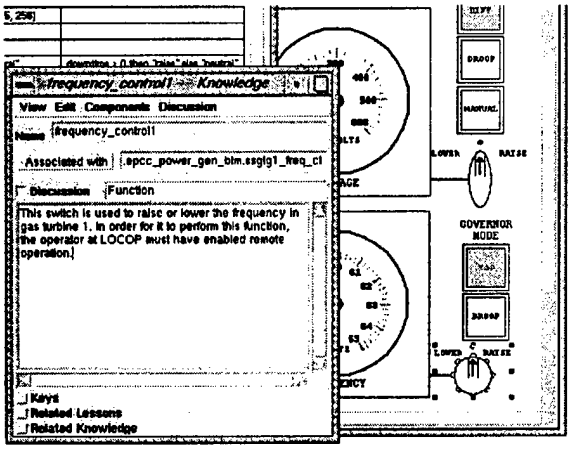


Figure 16. VIVIDS Knowledge Editor is used to create spoken instruction

Figure 17 shows the VE interface for knowledge exploration. A student can bring up an instructional user interface that consists of a menu of commands. Here, a student has entered the *Show information* mode by selecting that item on the command menu. The student then clicked on the frequency control knob at the lower right corner of the large vertical panel. This caused an information submenu to appear that presents the name of the object, its current state, and a list of the available discussion topics (Function and Operation, in this case). If the student selects the Function topic, VIVIDS sends the text on that topic (which was entered in the Knowledge Editor, shown in Figure 16) to TrishTalk so that it can be read aloud.

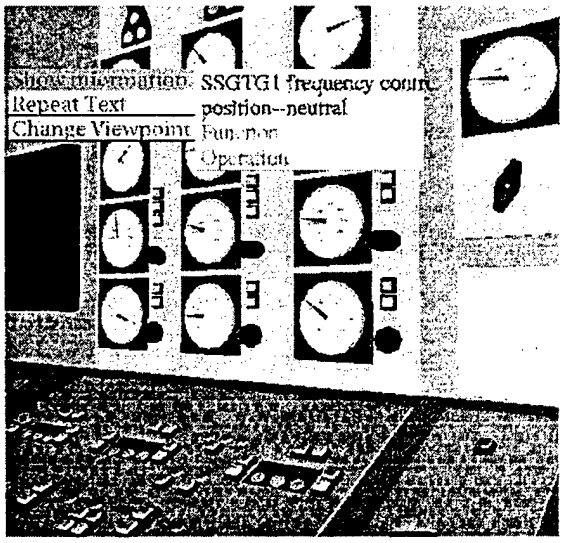


Figure 17. Immersed students can access authored knowledge as an aid to instruction

D.2.2.5 Team Training Features

Features have been added to VIVIDS to make it possible to direct certain elements of instruction to particular students in a team training task. In a VIVIDS lesson, actual interactions with students are controlled by elementary *lesson steps.* There are 24 types of these, including highlighting simulation objects, requiring an action, triggering a simulation event, presenting text, asking a question with a menu of answers, and so on. Authors can specify to whom most of these lesson steps should be directed. An example of a type of lesson step that cannot be presented to a particular student is a *Set Configuration*, which restores the entire simulation to some previously authored state. For those lesson steps that can be directed to particular students, the default is to direct the step to all the students.

For those types of lesson steps that require the manipulation of a simulation object, such as a switch, knob, or lever, if a participant is specified, then only that participant will actually be able to carry out the action. Attempts by other students to perform the required action will be blocked. This system gives authors a good deal of flexibility in determining how a team training lesson will be presented

D.2.3 Pedagogical Agents

This section provides a technical view of the Steve component of the Training Studio.

The VET project developed the Soar Training Expert for Virtual Environments (Steve) architecture for pedagogical agents. Steve has been used to create animated pedagogical agents that can monitor students performing tasks, demonstrate tasks, and answer questions. These agents appear in the virtual environment as virtual human figures, allowing them to participate as team members and engage in face-to-face dialogs with students. Such animated pedagogical agents are a natural means for delivering instruction in virtual environments.

D.2.3.1 Motivation

Virtual reality creates an opportunity for a new breed of computer tutor: the tutor can appear as an autonomous, animated agent that cohabits the virtual world with students and other agents. Such a *pedagogical agent* provides two key

advantages. First, the agent and student can carry on a face-to-face tutorial dialog, situated in the virtual world. Unlike previous disembodied computer tutors, the agent can demonstrate actions, use locomotion, gaze, and deictic gestures to guide the student's attention, and use many of the nonverbal cues that people use to regulate their conversations. Second, such agents can support team training; in addition to serving as tutors for students playing roles in a team, they can also play the roles of missing team members, allowing students to practice team tasks even when their human teammates are unavailable. To explore this new breed of computer tutor, we developed a pedagogical agent named Steve (Soar Training Expert for Virtual Environments).

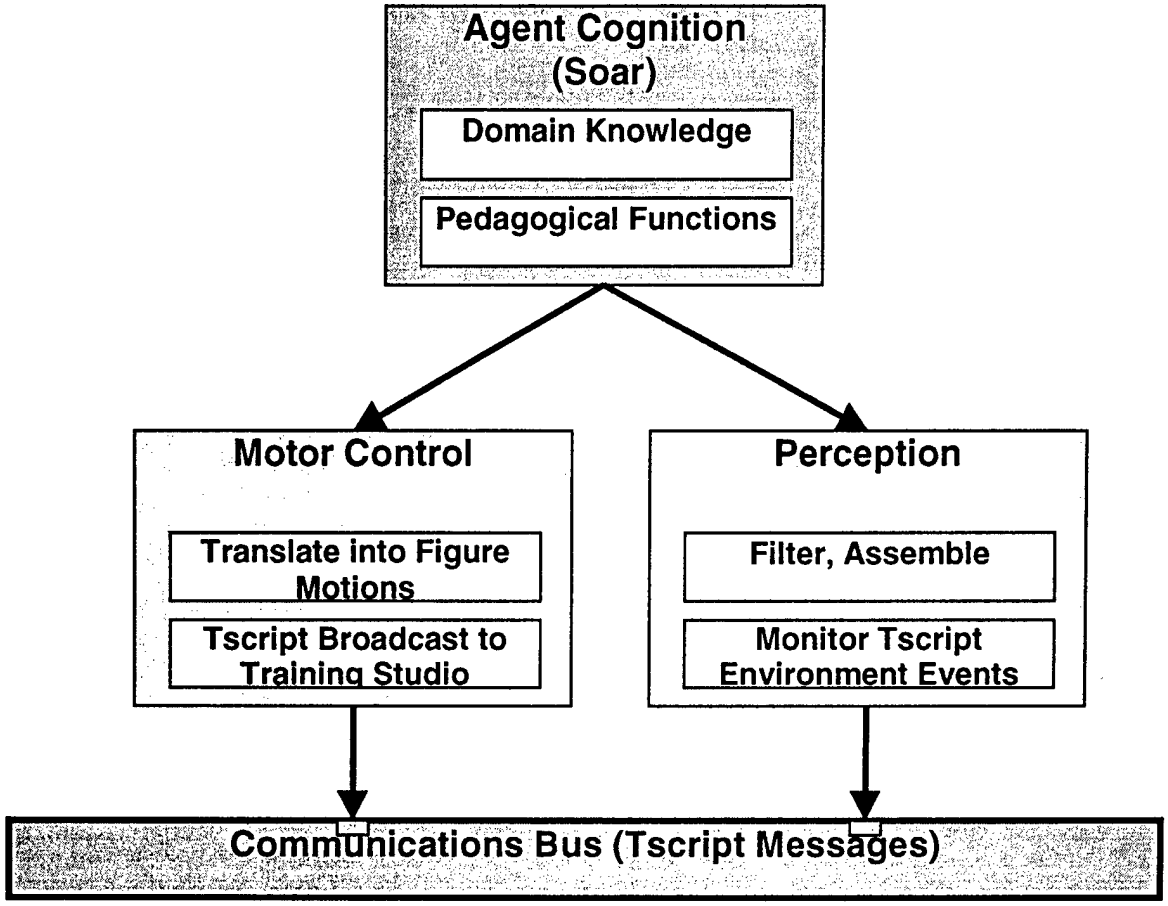
Steve integrates methods from three primary research areas: intelligent tutoring systems, computer graphics, and agent architectures. This novel combination results in a unique set of capabilities. Steve has many pedagogical capabilities one would expect of an intelligent tutoring system. For example, he can inform students when they make mistakes, and he can answer questions such as "What should I do next?" and "Why?" However, unlike previous intelligent tutoring systems, Steve appears as an animated human character in the virtual world, supporting a rich interaction with students. Moreover, Steve's agent architecture allows him to robustly handle a dynamic virtual world, potentially populated with people and other agents; he continuously monitors the state of the virtual world, always maintaining a plan for completing the current task, and revising the plan to handle unexpected events.

The following sections summarize Steve's architecture and capabilities. More technical details are available in our publications (Rickel and Johnson 1997, Johnson et. al. 1998).

D.2.3.2 Steve's Architecture

Like many other autonomous agents that deal with a real or simulated world, Steve consists of separate modules for perception, decision making, and motor control (see Figure 18). The perception module monitors messages from the communication bus and identifies events that are relevant to Steve, such as actions taken in the virtual world by people and agents and changes in the state of the virtual world. The cognition module, which is built on top of Soar (Laird et al. 1987; Newell 1990), interprets the input it receives from the perception module, chooses appropriate goals, constructs and executes plans to achieve those goals, and sends out motor commands to interact with the world. The motor control module decomposes the motor commands into a sequence of lower level commands that are sent out to the other VET components (e.g., Vista, VIVIDs, and TrishTalk) via the communication bus.

Figure 18. Steve's architecture consists of world state perception, motor control, and cognition blocks



The other components of the VET system provide a rich perceptual environment for Steve. Currently, his perception includes the state of the virtual world (objects and their attributes, updated by VIVIDs), actions taken by students and other agents, the location of students and agents, the set of objects within the student's field of view (updated by that student's Vista), human and agent speech (including messages when someone's speech begins and ends, along with the content of the utterance), and time. The perception module uses the messages from other VET components to maintain a coherent snapshot of the state of the virtual world, which it passes to the cognition module about 10 times per second.

D.2.3.3 Steve's Cognitive Capabilities

Steve's cognition module is organized into three layers. At the lowest layer lies Soar. Soar was designed as a general model of human cognition, so it provides a number of features that support the construction of intelligent agents, such as a frame-based representation of working memory, a production rule representation of long term memory, a decision cycle that includes input and output with an external world, a truth maintenance system, and automatic sub-goaling and chunking. However, Soar does not provide built-in mechanisms for particular cognitive skills, such as demonstration, explanation, and question answering. Therefore, one of our main tasks in building Steve was to design a layer of domain-independent pedagogical capabilities such as these on top of the Soar architecture. These capabilities are implemented as Soar production rules that can be applied to any domain. The final layer in Steve's cognition module is his knowledge of a particular domain, such as shipboard procedures. Given appropriate task knowledge for a particular domain, Steve uses his general pedagogical capabilities to teach that knowledge to students. Thus, our layered

approach to Steve's cognition module allows him to be used in a variety of domains; each new domain requires only new task knowledge without any modification of Steve's abilities as a teacher.

Steve's main objective is to teach students how to perform physical, procedural tasks, such as operating and repairing equipment. Thus, Steve needs a representation of the appropriate procedural knowledge for any domain that he must teach. Intelligent tutoring systems typically represent procedural knowledge in one of two ways. Some, notably those of Anderson and his colleagues (Anderson et al. 1995), use detailed cognitive models built from production rules. Other systems (Rickel 1988) use a declarative representation, usually some variant of a procedural network representation (Sacerdoti 1977) specifying the steps in the procedure and their ordering. Production rule models provide a more flexible ontology at a price: they are laborious to build and difficult to maintain. In contrast, procedural network representations are more practical for domains like operation and maintenance of equipment; procedures may change frequently in such domains, so it must be easy for domain experts or course authors to represent procedures, examine them, and change them when necessary. For these reasons, Steve uses a procedural network (plan) representation for domain tasks.

Steve's representation is not uncommon in the AI planning community (Russell and Norvig 1995), although, as we will discuss, it differs in important ways from representations in other tutoring systems. First, each plan consists of a set of steps, each of which is either a primitive action (e.g., press a button) or a composite action (i.e., a sub-plan). Composite actions give plans a hierarchical structure. Second, there may be ordering constraints among the steps. Finally, the rationales for the steps in the plan are represented as a set of causal links (McAllester and Rosenblitt 1991); each causal link specifies that one step in the plan achieves a goal that is a precondition for another step (or for termination of the task). For example, pulling out a dipstick achieves the goal of exposing the level indicator, which is a precondition for checking the oil level.

To teach a student how to perform domain tasks, Steve and the student practice the tasks together. All of Steve's instruction and assistance is situated in the performance of domain tasks; other types of instruction, like familiarization lessons, are handled by VIVIDS. Ideally, students should learn to apply standard procedures to a variety of situations, and they should learn the rationale behind steps in the procedure. Our goal is to support the apprenticeship model of learning (Collins et al. 1989). This requires two capabilities: Steve must be able to demonstrate and explain tasks, and he must be able to monitor students performing tasks, providing assistance when it is needed.

Whether Steve is demonstrating a task or monitoring the student, he must maintain a plan for completing the task. The plan allows Steve to choose the next appropriate action and, if asked, to explain the role of that action in completing the task. Steve's plans should follow standard domain procedures as much as possible. However, in order to handle dynamic environments, possibly containing other people and agents, Steve must be able to adapt those procedures to handle unexpected events (e.g., equipment failures, student errors, and teammate errors). Moreover, he must do so quickly, since he and the student are collaborating on the task in real time.

To satisfy these criteria, Steve uses a novel combination of task decomposition planning (Sacerdoti 1977) and partial-order planning (Weld 1994). Task decomposition planning is used to create a general model of the task (represented as a hierarchical plan). Partial order planning is used to decide which steps in the task model are required, in the current situation, to complete the task; this plan is updated whenever the state of the virtual world changes. This approach is efficient, and it forces Steve to follow standard procedures as much as possible, yet it still allows him to adapt the plan to unexpected events: Steve naturally re-executes parts of the plan that get unexpectedly undone, and he skips over parts of the plan that are unnecessary because their goals were serendipitously achieved. Thus, unlike videos or scripted demonstrations, Steve can adapt domain procedures to the state of the virtual world.

The causal links in Steve's representation of procedural knowledge are an important source of power, yet they have not been used in other tutoring systems. Causal links allow Steve to construct and revise plans, because they represent how each step contributes to achieving the end goals of a task. Previous tutoring systems based on procedural net representations,

on the other hand, only represent steps and ordering constraints. Without causal links, these systems are incapable of adapting procedures to unexpected situations. Causal links also allow Steve to explain to students the rationale for his actions and recommendations. Other procedural net tutors cannot automatically generate situation-specific explanations this way. Neither can tutors based on production rules; in those tutors, the relationships among actions are implicit in the rules.

Steve's collaboration with a student on a task is not rigid; he carries on a mixed-initiative dialog with the student, and can gracefully shift between demonstrating the task and monitoring the student performing the task. To ensure coherence when demonstrating a task, Steve maintains a dialog focus stack and dynamically selects cue phrases (e.g., "first" and "next") to indicate the relationship between a new step and the previous one. When demonstrating a step, Steve typically moves to the appropriate location, describes the step while pointing to the relevant object, performs the step, and describes any relevant results. The student can always interrupt a demonstration and ask to finish the task himself. When monitoring the student, Steve nods in acknowledgment of correct actions and indicates when the student has made an error. The student can ask Steve what to do next and why, and can also ask Steve to show him how to do it. Throughout this dialog, Steve makes appropriate use of gaze to regulate the conversation. To support after-action review, Steve uses Johnson's Debrief system (Johnson 1994) to maintain an episodic memory of his actions; after the task, if the student asks Steve why he did something, Steve recalls the situation and explains his rationale.

D.2.3.4 Steve's Motor Control

Steve's motor control module receives motor commands from the cognition module and decomposes them into lower-level commands that are sent to other VET components via the communication bus. The motor control module accepts a variety of commands: speak to someone, move to an object, look at something, nod or shake the head, point at an object, manipulate an object (in various ways), and change facial expression. The motor control module determines how these commands are realized in the virtual world; Steve's entire body can be replaced with a new one by simply re-implementing the motor control module.

We have experimented with several bodies for Steve. The current version represents a full upper body of a human figure. We created the graphical models and animation control from scratch (although the head was derived from a public domain version of the Jack human figure developed at the University of Pennsylvania). Although software for controlling human figures is available from several universities, none was suitable for our purposes. For example, the Jack software (Badler et al. 1993) could not be used outside its own browser (i.e. could not be run inside Vista), and its API for control by external programs (such as Steve) was not sufficiently developed (Jack was initially designed for control by humans via menus). The animation control code we developed has proven to be efficient, robust, and natural looking, and it has given us the opportunity to experiment with different functionality to support our research.

Control over Steve's body is split into two pieces. Steve's motor control module controls gross movement, while fine-level control (e.g., moving Steve's arm to an object or having Steve's gaze track a moving object) is handled by code running as a shared library within each Vista Viewer. The shared library, which was developed at ISI, has its own API, providing another layer of modularity.

Although the shared library was developed for control by Steve, it proved useful for VIVIDS as well. An optional feature of the structured lessons offered by VIVIDS is the use of a directable version of Steve's body under the control of the VIVIDS lesson step routines. This makes it possible for an author to quickly build a structured lesson that uses Steve to remediate certain student errors or to carry out demonstrations. If the author chooses *not* to use Steve, remediations and demonstrations are carried out by graphically highlighting objects and then requiring the student to select them. After the student touches the object, the lesson continues. In the case of an action demonstration, the simulation effects of the required action are displayed once the student has selected the relevant control. Use of Steve for such demonstrations provides additional information that is not available in a conventional VIVIDS demonstration.

D.2.3.5 Team Training

Agents like Steve are especially useful for team training, where they can serve not only as tutors for individual students in the team but also as missing team members. Steve's ability to perform actions in the virtual world allows his teammates to follow his actions. His ability to use speech synthesis and speech recognition allows him to communicate with his teammates. His ability to adapt to unexpected situations allows him to robustly operate in a virtual world with other people and agents. Thus, although Steve was originally designed for one-on-one tutoring, we were able to extend him to support team training with relatively few extensions.

To support team training, we generalized Steve's task model representation. In addition to specifying steps, ordering constraints, and causal links, each team task model also has a set of roles (e.g., electrical operator and propulsion operator). The task model specifies which role is responsible for each step in the task. Once students and agents are assigned to the roles in a task, the information in the task models allows the agents to determine the actions for which each team member is responsible. This approach would not work for tasks that require teammates to dynamically negotiate role assignments; fortunately, many real-world tasks, particularly in the military, have fixed role assignments.

Natural language communication is often critical to team coordination. To support this, we model speech acts as explicit actions in the task model. For example, one team member may communicate to another that a subtask is complete. When an agent's teammate (human or agent) says something, the agent interprets the utterance by comparing it to speech acts in the task model that are appropriate in the current situation. When a student says something inappropriate, the agent serving as the student's tutor is responsible for providing feedback. This approach works well when the possible utterances among teammates can be specified ahead of time. Again, although this rules out many ill-structured tasks, it is sufficient for many team tasks, particularly in the military, which have a prescribed set of utterances for which Steve agents can listen.

D.2.3.6 Authoring by Demonstration

One of our main objectives in designing Steve was that it should be easy to provide the knowledge he needs to teach a new domain. To meet this objective, we separated Steve's domain knowledge from his general pedagogical capabilities, and we ensured that he only relies on types of domain knowledge that a course author could easily provide and maintain. To further simplify the course author's job, we have developed tools that use machine learning to help automate the acquisition of domain task knowledge.

To teach Steve about a new task, the course author demonstrates the steps of the task in the virtual world. For each action in the demonstration, Steve notes the state of the virtual world before and after the action; this provides one example of the effects of the action on the virtual world. Steve also requires the author to describe the action via a text string; these text strings will form the basis for Steve's later instruction to students. At the end of the demonstration, Steve shows the author a list of changes in the state of the virtual world that resulted from the demonstration; the author distinguishes those that are the end goals of the task from those that are incidental side effects. Now, given this demonstration and understanding of the new task's goals, Steve must learn two things: he must learn the causal links (i.e., the causal dependencies among the actions, as described earlier), and he must learn the ordering constraints among steps (in case some steps can be done in any order).

To identify the causal links and ordering constraints, Steve experiments with variants of the author's demonstration. By systematically dropping different steps from the procedure and performing the resulting variant in the virtual world, Steve learns the preconditions and effects of each step, from which he can reconstruct the causal links and ordering constraints for the task. Steve's learning procedure is a novel variant of Mitchell's Version Space algorithm for inductive learning (Mitchell 1982).

After Steve has finished experimenting with the task, he presents his understanding of the task to the course author. At this point, the author can make any corrections to the task model in cases where Steve was unable to generate enough

examples to eliminate some possible dependencies among steps. Thus, through a combination of programming by example (Cypher 1993), learning by observation (Wang 1995), and learning by experimentation (Gil 1993), the author's task is reduced from providing the entire task description to simply providing a demonstration and making any necessary changes in Steve's resulting understanding.

For a more detailed description of this work, see (Angros et al. 1997) and (Johnson et al. 1998).

D.2.3.7 Related Work

Although Steve draws on a long line of research in intelligent tutoring systems, agent architectures, planning, and machine learning, no previous systems integrate his unique range of capabilities. The most closely related pedagogical agent for virtual reality was developed by Billinghurst and his colleagues (Billinghurst and Savage 1996; Billinghurst et al. 1996). Their agent inhabits a 3D, simulated nasal cavity, providing assistance in sinus surgery to medical students. However, their agent does not have an animated form, and it cannot adapt procedures to unexpected events. Lester and his colleagues are developing two animated pedagogical agents (Stone and Lester 1996; Lester et al. 1998), but their agents do not inhabit 3D virtual worlds; they appear as 2D characters floating on top of a 2D image of a virtual world. Also, they do not interact with a simulator, nor do they have any ability to construct or execute domain plans. The PPP Persona (Andre, Rist, and Mueller 1998) is a 2D animated agent that can combine speech and gestures to describe procedures for operating physical devices, but it cannot interact with a simulator, and it has no pedagogical capabilities except the ability to describe a procedure. None of these other agents supports team training.

E Summary

This section summarizes the Virtual Environments for Training effort, covering significant results, suggested courses of action, and future plans.

E.1 Significant Results

The VET project focused on immersed instruction, culminating with the successful development of the Training Studio prototype, an authorable system for team instruction in a virtual environment. Several key advances have resulted from the VET program:

- Pedagogical agents that can act as student mentors or team members during training
- Component-based instructional systems
- Authorable virtual environment interaction applicable across domains

In the executive summary of our 1994 VET proposal, we proposed to *develop an instructional system that integrates the design, development, delivery, and evaluation of training curricula with a comprehensive virtual environment.* Using our Training Studio system, authors can design and develop instruction for delivery to students, and evaluation of their performance, in an immersed virtual environment. Thus, the VET Training Studio is an example of an integrated, working VE system for constructing, managing, and interacting within virtual environments.

VET has advanced the state of the art in the use of intelligent agent technology for training in virtual environments. Steve is able interact with students in un-scripted training scenarios. He can act either as instructor or team member. By integrating intelligent agents (pedagogical agents) into virtual environments, we have significantly enhanced the value of such environments as vehicles for training.

VET has shown the feasibility for component based instructional authoring and delivery tools. Our separation of presentation and interaction (Vista), equipment model (VIVIDS), and human models (Steve) has shown that its possible to attain advanced instructional capabilities using a component-based approach.

VET has shown that immersed virtual environment interaction can be authored using standards for 3D geometry, constraints, and behavior, in a manner that is not specific to the VE software, and in a manner that can be used with advanced instructional components to achieve new types of interaction with the Student for instruction, such as those shown with Steve, and the general concept of Spatial Dialog.

E.2 Suggested Course of Action

The VET work has demonstrated the potential of virtual environments to provide effective team training at reduce cost. ONR should encourage further evaluation of the technology on fleet problems, in order to encourage the transition into Navy training practice.

Meanwhile, further research and development needs to be conducted on ways of integrating this technology into work environments via augmented reality techniques, combining pedagogical agent technology with state-of-the-art human figure technology such as Jack, and exploiting the tracking data in the virtual environment in order to perform quantitative assessments of student performance.

One novel research area uncovered during this effort is spatial dialog, where the spatial context of the user is used to conduct a meaningful spoken dialog with a computer. Spatial context includes direction of the person's gaze, the objects in their view, the objects they are touching, the objects they can touch, the objects they are near, the routes they can navigate, and their current spatial task. By using spatial context as part of understanding a user's speech, more natural language understanding capabilities can be achieved. Similarly, by using visual cues in the spatial context, the speech produced by the computer can be enhanced. Steve's actions to point out equipment, and to lead the way through the ship during instruction are examples of this. Prior to virtual environments, much of the spatial context was not available for speech dialog systems, and many projects continue to ignore spatial context in a world where we largely engage in spatial dialog with each other. An example of this is a an instructor pointing at a wrench, and telling the student to "Pick it up", another example is a waitress bringing a cart of deserts to a table and asking the people seated there "Which of these would you like?" There are many more complex examples that we handle in everyday life.

E.3 Future Plans

Within Lockheed Martin, the Advanced Technology Center will continue development of Vista. We plan to expand Vista's capabilities in three areas: mediated reality, where 3D scenes are over-layed on the real surrounding scene, enterprise integration, where CAD models are used for prototyping training and usage, and spatial dialog, where visual context is used to dramatically improve human-computer spoken dialog.

USC / Behavioral Technology Laboratories continues to build on the VIVIDS technologies under two funded research projects. The lessons learned in the VET project are driving the development of a newVivids architecture for simulation-centered tutors. The goal here is to develop an architecture and certain realized components of that architecture for the development and delivery of simulation-centered tutors in a very wide range of contexts. One application of the architecture will be a facility for authoring 2D simulation tutors for delivery on Java platforms, using a lightweight universal simulation player. A data-driven tutorial player is also under development. This tutor delivery system is designed to work in a wide variety of both 2D and VE-based simulation environments.

The pedagogical agent technology developed as part of the VET project has broad applicability potential both in immersive virtual environments and in more conventional desktop environments. USC / ISI has teamed with Intelligent Systems Technology, Inc. on an Air Force SBIR Phase II grant to productize the technology. Meanwhile, Steve provided the basis for USC / ISI's Adele pedagogical agent, designed for instructional use in the health sciences. USC plans to incorporate pedagogical agent technology into health science courses in the spring of 1999.

USC / Information Sciences Institute's AASERT grant is a three-year effort focusing on two areas: natural interaction with students, and natural action with instructors. Richard Angros, a Ph.D. student, developed an authoring interface called Diligent that allows instructors to author Steve's task knowledge by demonstration (Angros et al 1997). Using Diligent, a course author can demonstrate tasks to Steve. Diligent then directs Steve through a process of trying variations on the demonstrated task, in order to generalize from the demonstration examples. In the coming year Mr. Angros will perform usability evaluations of Diligent and write up and publish his work. Ben Moore, an undergraduate, has developed an interface for authoring the interface between Steve and the virtual environment, e.g., defining Steve's primitive manipulations of virtual objects. Our research in natural interaction with students has investigated the use of auditory feedback to facilitate student interaction with the virtual environment. In our continuing work, Steve will maintain a model of the ongoing dialog with students, in order to facilitate natural interaction between students and Steve agents.

F Bibliography

This bibliography is a list of cited articles, as well as a list of related works not specifically cited here, but of relevance to the VET work.

The Virtual Environments for Training web site has references to other sites in this subject area.

http://vet.parl.com/~vet/

The VET report web page has copies of some of the papers listed here, as well as links to USC/ISI and USC/BTL.

http://vet.parl.com/~vet/reports/

Angros, R., Johnson, W.L., & Rickel, J., *Agents that Learn to Instruct*, AAAI 1997 Fall Symposium Series: Intelligent Tutoring Systems Authoring Tools, Technical Report FS-97-01, November 1997, AAAI Press.

Ambros-Ingerson, J.A., and Steel, S., 1988. Integrating planning, execution and monitoring. In *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-88)*, pp. 83-88, San Mateo, CA, Morgan Kaufmann.

Anderson, J.R., Boyle, C.F., Corbett, A.T., and Lewis, M.W., 1990. Cognitive modeling and intelligent tutoring. *Artificial Intelligence* (42), pp. 7-49.

Badler, N., Phillips, C., and Webber, B., 1993. *Simulated Agents and Virtual Humans*, Oxford University Press.

Barfield, W., Furness, T. A., Virtual Environments and Advanced Interface Design, Oxford University Press, 1995. P. 11.

Barrus, J.W., Waters, R.C., and Anderson, D.B., 1996. Locales and beacons: Efficient and precise support for large multi-user virtual environments. *Proceedings of the IEEE Virtual Reality Annual International Symposium*, pp. 204-213. IEEE Computer Society Press.

Billinghurst, M., and Savage, J., 1996. Adding intelligence to the interface. *Proceedings of the IEEE Virtual Reality Annual International Symposium*, pp. 168-175. IEEE Computer Society Press.

Blumberg, B.M. and Galyean, T.A., 1995. Multi-level direction of autonomous creatures for real-time virtual environments. *SIGGRAPH 95 Conference Proceedings*, pp. 47-54.

Brooks, R.A., 1986. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation* 2(1): 14-23.

Calvin, J. et al, 1993. The SIMNET virtual world architecture. *Proceedings of the IEEE Virtual Reality Annual International Symposium*, pp. 450-455. . IEEE Computer Society Press.

Coller, L. D., Pizzini, Q. A., Wogulis, J., Munro, A. & Towne, D. M. , 1991. Direct manipulation authoring of instruction in a model-based graphical environment. In L. Birnbaum (Ed.), *The international conference on the learning sciences: Proceedings of the 1991 conference*, Evanston, Illinois: Association for the Advancement of Computing in Education.

Collins , A., 1989. Cognitive apprenticeship: teaching the crafts of reading, writing, and mathematics. In Resnick, L.B., ed., *Knowing, Learning, and Instruction: Essays in Honor of Robert Glaser*, pp. 453-494, Lawrence Erlbaum Associates.

Cutler, L. D., Frohlich, B., Hanrahan, P., Two-Handed Direct Manipulation On The Responsive Workbench, Symposium on Interactive 3D Graphics, 1997.

Cypher, A., ed., 1993. *Watch What I Do: Programming by Demonstration.* MIT Press, Cambridge, MA.

Dewey, J., 1939. Fundamentals of educational process. *Intelligence in the Modern World: John Dewey's Philosophy.* Edited by Joseph Ratner. New York: Random House, Inc.

de Jong, T., van Joolingen, W., Scott, D., deHoog, R., Lapied, L., Valent, R., SMILSLE: system for multimedia integrated simulation learning environments. In T. de Jong and L. Sarti (Eds.) *Design and production of multimedia and simulation based learning material*, Dordrecht: Kluwer Academic Publishers, 1994.

Durlach, N.I., and Mavor, A.S., eds., 1995. *Virtual Reality: Scientific and Technological Challenges.* National Academy Press, Washington D.C.

Firby, R.J., 1994. Task networks for controlling continuous processes. In *Proceedings of the Second International Conference on AI Planning Systems.*

Fogg., B.J., and Moon, Y., 1994. Computer as teammate: effects on user attitude and behavior. *Proceedings of Lifelike Computer Characters '94,* p. 54, Microsoft Research, Snowbird, UT.

Grant, F., McCarthy, L., Pontecorvo, M. & Stiles, R. *Training in Virtual Environments.* Conference on Intelligent Computer-Aided Training. Houston, TX: NASA Johnson Space Center, November 1991.

Gabbard, J. L. Hix, D., *A Taxonomy of Usability Characteristics in Virtual Environments,* Virginia Polytechnic Inst. & State Univ. Technical Report, Nov 1997. See http://csgrad.cs.vt.edu/~jgabbard/ve/taxonomy/

Guiard, Yves. *Symmetric Division Of Labor In Human Skilled bimanual action: the kinematic chain as a model,* The Journal of Motor Behaviour, 19(4):486-517, 1987.

Hall, C., Stiles, R., Horwitz, C., *Virtual Reality for Training: Evaluating Knowledge Retention.* Accepted for 1998 IEEE Virtual Reality Annual International Symposium (VRAIS '98), March 1998, Atlanta GA.

Hayes-Roth, B. and van Gent., R., 1997. Story Making with Improvisational Puppets, *Proceedings of the First International Conference on Autonomous Agents,* 1997.

Hill, R.W. and Johnson, W.L., 1995. Situated Plan Attribution, *Journal of Artificial Intelligence in Education* 6(1), pp. 35-67.

Hollan, J. D., Hutchins, E. L., and Weitzman, L., 1984. STEAMER: and interactive inspectable simulation-based training system, *AI Magazine* 5(2), pp. 15-27.

Horwitz, C. D., Fleming, J., Regian, W., Stiles, R, *Software tools for embedding principled instruction in virtual environment simulations,* Proc. IMAGE 96 Conf, Scottsdale, AZ, June 1996.

Gil, Y., 1993. Efficient Domain-Independent Experimentation. USC / ISI technical report ISI/RR-93-337. Also appears in the *Proceedings of the Tenth International Conference on Machine Learning.*

Johnson, W.L., 1986. *Intention-Based Diagnosis of Novice Programming Errors,* Morgan Kaufmann, Menlo Park, CA.

Johnson, W.L., 1994. Agents that learn to explain themselves. *Proceedings of the Twelfth National Conference on Artificial Intelligence,* pp. 1257-1263. AAAI Press, Menlo Park, CA.

Johnson, W.L. and J. Rickel, "Intelligent Tutoring in Virtual Environment Simulations," *ITS '96 Workshop on Simulation-Based Training Technology,* June 1996.

Johnson, W. L., Rickel, J., Stiles, R., and Munro, A., *Integrating Pedagogical Agents into Virtual Environments.* Presence Journal 7(6) Dec 1998., MIT Press.

Kotani, A. and Maes, P., 1994. Guide agents for virtual environment. *Proceedings of Lifelike Computer Characters '94,* Microsoft Research, Snowbird, UT, p. 59.

Laird, J.E., Newell, A., and Rosenbloom, P.S., 1987. Soar: An architecture for general intelligence. *Artificial Intelligence* 33(1), pp. 1-64.

Loftin, R.B., and Kenney, P., 1995. Training the Hubble space telescope flight team. *IEEE Computer Graphics and Applications* 15(5): 31-37.

McCarthy, L., Stiles, R., Pontecorvo, M. & Grant, F. *Spatial Considerations for Instructional Development in a Virtual Environment.* 1993 Conference on Intelligent Computer-Aided Training and Virtual Environment Technology. Houston, TX: NASA Johnson Space Center, May 1993.

McCarthy, L., Stiles, R., Rickel, J. Johnson, L., *Human-Systems Interaction for Immersed Training,* In Press, Virtual Reality Journal, Springer Verlag.

McCarthy, L., Stiles, R., Rickel, J. Johnson, L., *Enabling Team Training in Virtual Environments*, Proc. Collaborative Virtual Environments, Manchester, U.K., June 1998.

Mine, Mark R., Brooks, Frederick P., Sequin, Carlo H. *Moving Objects In Space: Exploiting Proprioception In Virtual Environment Interaction*, Proceedings of SIGGRAPH '97, Los Angeles, CA, Aug. 1997.

Mitchell, T.M., 1982. Generalization as search. *Artificial Intelligence* 18: 203-266.

Munro, A., Johnson, M.C., Surmon, D.S., and Wogulis, J.L., 1993. Attribute-centered simulation authoring for instruction, *Proceedings of the AI-ED 93 World Conference of Artificial Intelligence in Education*, pp. 82-89, Edinburgh, Scotland.

Munro, A. Authoring interactive graphical models. In T. de Jong, D. M. Towne, and H. Spada (Eds.), *The Use of Computer Models for Explication, Analysis and Experiential Learning*. Springer Verlag, 1994.

Munro, A. *RIDES QuickStart*, Los Angeles: Behavioral Technology Laboratories, University of Southern California, 1995.

Munro, A. and Pizzini, Q. A. *RIDES Reference Manual*, Los Angeles: Behavioral Technology Laboratories, University of Southern California, 1995.

Munro, A., Johnson, M.C., Pizzini, Q.A., Surmon, D.S., and Wogulis, J.L, A Tool for Building Simulation-Based Learning Environments, in *Simulation-Based Learning Technology Workshop Proceedings, ITS'96*, Montreal, Quebec, Canada, June 1996.

Newell, Allen, *Unified Theories of Cognition.*, Harvard University Press, Cambridge, MA, 1990.

Pizzini, Q.A., Munro, A., Wogulis, J.L., and Towne, D.M., The cost-effective authoring of procedural training, in *Architectures and Methods for Designing Cost-Effective and Reusable ITSs* Workshop Proceedings, ITS'96, Montreal, Quebec, Canada, June 1996.

Regian, J.W, Shebilske, W., and Monk, J., 1992. A preliminary empirical evaluation of virtual reality as an instructional medium for visual-spatial tasks. *Journal of Communication* 42(4): 136-149.

Rich, C., 1995. Diamond Park demonstration. *IJCAI Workshop on AI and Entertainment*, Montreal, Que.

Rickel, J., 1988. An intelligent tutoring framework for task-oriented domains. In *Proceedings of the International Conference on Intelligent Tutoring Systems*, Montreal, Canada.

Rickel, J. and W. Lewis Johnson, *Integrating pedagogical capabilities in a virtual environment agent*, in *Proceedings of the First International Conference on Autonomous Agents*, February 1997.

Renze, K.J. and J. H. Oliver, *Generalized Surface and Volume Decimation for Unstructured Tessellated Domains, Proceedings of the IEEE 96 VRAIS*, March 1996, Santa Clara, CA, pp. 111-121.

Rohlf, J. & J. Helman. *IRIS Performer: A High Performance Multiprocessing Toolkit for Real-time 3D Graphics*, Proceedings of SIGGRAPH '94, Orlando FL, Aug.1994.

Russell, Stuart and Peter Norvig, *Artificial Intelligence: A Modern Approach*, Prentice Hall, Englewood Cliffs, NJ, 1995.

Sacerdoti, E., 1977. *A Structure for Plans and Behavior.* Elsevier North-Holland, New York.

Self, J. 1995. The ebb and flow of student modeling, *Proceedings of the International Conference on Computers in Education (ICCE '95)*: 40-40h.

Spensley, F. and Elsom-Cook, M. Generating domain representations for ITS. In D. Bierman, J. Breuker, and J. Sandberg (Eds.), *the proceedings of the fourth international conference on artificial intelligence and education.* Amsterdam: IOS, 1989, 276-280.

Stansfield, S.A., 1994. A distributed virtual reality simulation system for situational training. *Presence* 3(4): 360-366.

Stansfield, S.A., Miner, N., Shawver, D., and Rogers, D., 1995. An application of shared virtual reality to situational training. In *Proceedings of the IEEE Virtual Reality Annual International Symposium,* pp. 156-161.

Sterling, B., 1993. War is virtual hell. *Wired* 1(1), pp. 46-51.

Stiles, R., L. McCarthy, and M. Pontecorvo, "Training studio: a virtual environment for training," *1995 Workshop on Simulation and Interaction in Virtual Environments (SIVE95)* Iowa City, IW: ACM Press, July 1995. See http://vet.parl.com/~vet/studio/tstudio_sive95_ToC.html

Stiles, R., *Human-Systems Interaction for Immersed Training* NASA Workshop on Advanced Training Technologies and Learning Environments, NASA Langley, Hampton, VA, March 1999.

Stiles, R., Tewari, S., and Mehta, M., McCarthy, L., *Adapting VRML for Immersed Free-form Manipulation* Accepted for VRML 98, Third Symposium on the Virtual Reality Modeling language, Monterey, CA February 1998.

Stiles, R., Tewari, S., and Mehta, M., *Adapting VRML 2.0 for Immersive Use.* In Proc. VRML 97 Second Symposium on the Virtual Reality Modeling language, Monterey, CA February 1997.

Stiles., R. et al. *Virtual Environments for Shipboard Training.* In Proc., Intelligent Ships Symposium II, The American Society of Naval Engineers, Philadelphia, PA, November 1996.

Stiles, R. & Pontecorvo, M. *Lingua Graphica: A Visual Language for Virtual Environments.* Proceedings of the IEEE International Workshop on Visual Languages. Seattle, WA: IEEE Press, September 1992.

Tambe, M., Johnson. W.L., and Shen, W.-M., 1997. Adaptive agent tracking - A preliminary report. *International Journal of Human-Computer Systems*, accepted for publication.

Tambe, M., Johnson, W.L., Jones, R.M., Koss, F., Laird, J.E., Rosenbloom, P.S., and Schwamb, K., 1995. Intelligent agents for interactive simulation environments, *AI Magazine* (6)1, pp. 15-39.

Tate, D.L., L. Sibert, F.W. Williams, T. King, and D.W. Hewitt, *Virtual environment firefighting/ship familiarization feasibility tests aboard the ex-USS Shadwell*, NRL Ltr Rpt 6180/0672A.1, Oct 1995.

Towne, D. M. A generalized model of fault-isolation performance. In *Proceedings, Artificial Intelligence in Maintenance: Joint Services Workshop*, 1984.

Towne, D. M. & Munro, A. The intelligent maintenance training system. In J. Psotka, L. D. Massey, and S. A. Mutter (Eds.), *Intelligent tutoring systems: Lessons learned* (479-530). Hillsdale, NJ: Erlbaum, 1988.

Towne, D. M., Munro, A., Pizzini, Q. A., Surmon, D. S., Coller, L. D., & Wogulis, J. L. Model-building tools for simulation-based training. *Interactive Learning Environments*, 1991, **1**, 33-50.

Towne, D. M. & Munro, A. Simulation-based instruction of technical skills. *Human Factors*, 1991, 33, 325-341.

Wang, X., 1995. Learning by observation and practice: An incremental approach for planning operator acquisition. *Proceedings of the 12th International Conference on Machine Learning.*

Webber, B. and Badler, N., 1993. Virtual interactive collaborators for simulation and training. *Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation*, pp. 199-205, Institute for Simulation and Training, Orlando, FL.

Weld , D.S., 1994. An introduction to least commitment planning. *AI Magazine*, 15(4):27-61.

Wenger, E., 1987. *Intelligent Tutoring Systems.* Morgan Kaufmann, Menlo Park, CA.

Williams, M. D., Hollan, J. D., and Stevens, A. L. An overview of STEAMER: an advanced computer-assisted instruction system for propulsion engineering. *Behavior Research*

methods and Instrumentation, 1981, 13, 85-90.